

# Working with a project

4D projects are created and developed using the **4D** application, which provides a comprehensive Integrated Development Environment (IDE). **4D Server** can also create new, empty projects.

Multi-user development is managed via standard **source control** repository tools (Perforce, Git, SVN, etc.), which allow developers to work on different branches, and compare, merge, or revert modifications.

## Creating a project

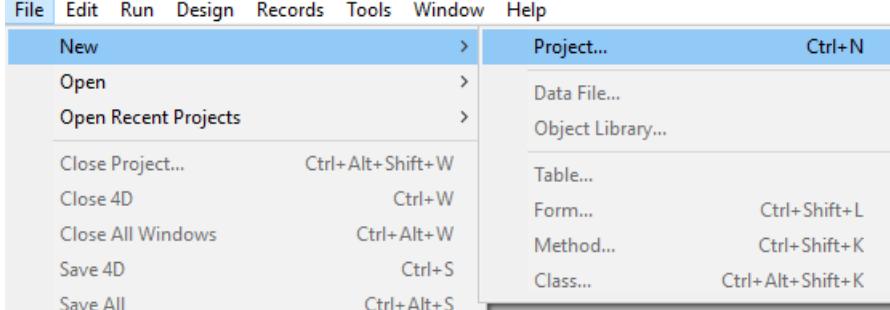
New 4D application projects can be created from **4D** or **4D Server**. In any case, project files are stored on the local machine.

To create a new project:

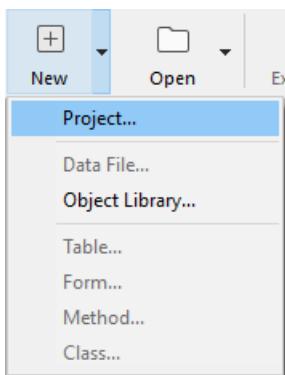
1. Launch 4D or 4D Server.

2. Do one of the following:

- Select **New > Project...** from the **File** menu:



- (4D only) Select **Project...** from the **New** toolbar button:



A standard **Save** dialog appears so you can choose the name and location of the 4D project's main folder.

3. Enter the name of your project folder and click **Save**. This name will be used:

- as the name of the entire project folder,
- as the name of the .4DProject file at the first level of the ["Project" folder](#).

You can choose any name allowed by your operating system. However, if your project is intended to work on other systems or to be saved via a source control tool, you must take their specific naming recommendations into account.

When you validate the **Save** dialog, 4D closes the current project (if any), creates a project folder at the indicated location, and puts all files needed for the project into it. For more information, refer to [Architecture of a 4D Project](#).

You can then start developing your project.

## Opening a project

To open an existing project from 4D:

1. Do one of the following:
  - Select **Open/Local Project...** from the **File** menu or the **Open** toolbar button.
  - Select **Open a local application project** in the Welcome Wizard dialog

The standard Open dialog appears.

2. Select the project's `.4dproject` file (located inside the ["Project" folder of the project](#)) and click **Open**.

By default, the project is opened with its current data file. Other file types are suggested:

- *Packed project files*: `.4dz` extension - deployment projects
- *Shortcut files*: `.4DLink` extension - store additional parameters needed for opening projects or applications (addresses, identifiers, etc.)
- *Binary files*: `.4db` or `.4dc` extension - legacy 4D database formats

## Options

In addition to standard system options, the *Open* dialog in 4D provides two menus with specific options that are available using the **Open** button and the **Data file** menu.

- **Open** - opening mode of the project:
  - **Interpreted** or **Compiled**: These options are available when the selected project contains both [interpreted and compiled code](#).
  - **Maintenance Security Center**: Opening in secure mode allowing access to damaged projects in order to perform any necessary repairs.
- **Data file** - specifies the data file to be used with the project. By default, the **Current data file** option is selected.

## Project opening shortcuts

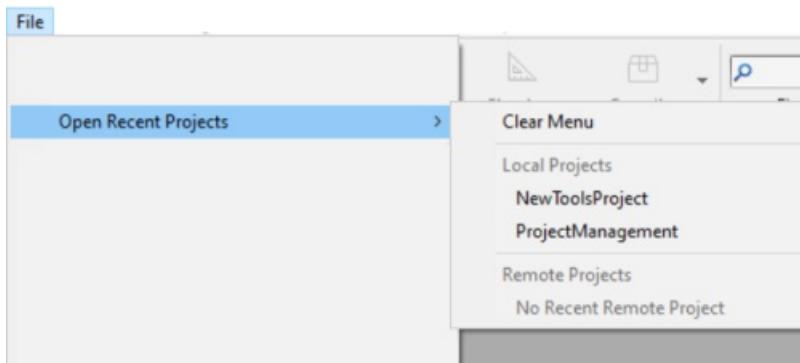
4D offers several ways to open projects directly and bypass the Open dialog:

- via menu options:
  - *Menu bar - File > Open Recent Projects / {project name}*
  - *4D Tool bar* - Select the project from the menu associated with the **Open** button
- via preferences:
  - Set the **At startup** general preference to **Open last used project**.
- using a `.4DLink` file.

## Opening a Project with a 4DLink file

You can use a [.4DLink file](#) to launch the 4D application and open the target 4D project. There are two ways to do this:

- double-click or drag and drop the `.4DLink` file onto the 4D application
- go to **File > Open Recent Projects** and select a project



A `.4DLink` file of "remote project" type can be copied and used on several machines.

It's also possible to select a `.4DLink` file in the 4D and 4D Server opening dialog box (opening local project only).

## About 4DLink Files

Files with the `.4DLink` extension are XML files that contain parameters intended to automate and simplify opening local or remote 4D projects.

`.4DLink` files can save the address of a 4D project as well as its connection identifiers and opening mode, saving you time when opening projects.

4D automatically generates a `.4DLink` file when a local project is opened for the first time or when connecting to a server for the first time. The file is stored in the local preferences folder at the following location:

- Windows 7 and higher: C:\Users\UserName\AppData\Roaming\4D\Favorites vXX\
- OS X: Users/UserName/Library/Application Support/4D/Favorites vXX/

XX represents the version number of the application. For example, "Favorites v19" for 4D v19.

That folder is divided into two subfolders:

- the **Local** folder contains the `.4DLink` files that can be used to open local projects
- the **Remote** folder contains the `.4DLink` files of recent remote projects

`.4DLink` files can also be created with an XML editor.

4D provides a DTD describing the XML keys that can be used to build a `.4DLink` file. This DTD is named database\_link.dtd and is found in the ¥Resources¥DTD¥ subfolder of the 4D application.

## File saving

When working on a project in 4D, you can use built-in 4D editors to create, modify, or save structure items, methods, forms, etc. Modifications are saved to disk when you

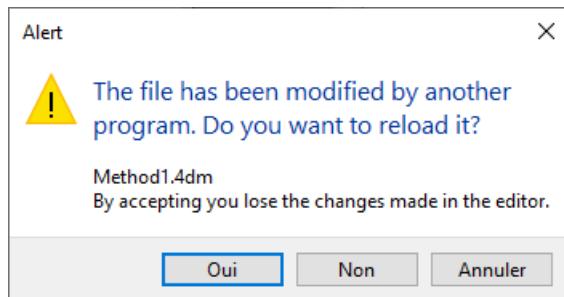
select a **Save** menu item, or when the editor's window loses or gets the focus.

Since the editors use files on the disk, potential conflicts could happen if the same file is modified or even deleted from different locations. For example, if the same method is edited in a Code Editor window *and* in a text editor, saving both modifications will result in a conflict.

The 4D development framework includes a file access manager to control concurrent access:

- if an open file is read-only at the OS level, a locked icon is displayed in the editor:  

- if an open file is edited concurrently from different locations, 4D displays an alert dialog when trying to save the changes:



- **Yes**: discard editor changes and reload the modified version
- **No**: save changes and overwrite the other version
- **Cancel**: do not save

This feature is enabled for all built-in 4D editors (Structure, Form, Method, Settings, and Toolbox).

# Release Notes

## 4D v20

Read [What's new in 4D v20](#), the blog post that lists all new features and enhancements in 4D v20.

### Index rebuild warning

4D v20 includes an ICU library update (see below) which will force an automatic rebuild of indexes of type alpha, text, and object. Depending on the size of the data file, this operation can take a while and may require to be planned.

### Highlights

- 4D Server automatically integrates multiple journals: [Automatic restore](#).
- [IMAP Transporter Class](#): `.getBoxInfo()` returns *id*, `.selectBox()` returns *id*, *flags* and *permanentFlags*, `.addFlags()` and `.removeFlags()` support custom keywords.
- New [WebSocketServer](#) and [WebSocketConnection](#) classes to create and manage WebSocket connections from 4D.
- Support of [property](#) keyword in user class definitions.
- New functions to lock/unlock the datastore: `.flushAndLock()`, `.locked()`, `.unlock()`.
- New `.at()` function in the Entity selection class.
- New functions in the Collection class: `.at()`, `.first()`, `.flat()`, `.flatMap()`, `.includes()`, `.last()`, `.reduceRight()`.
- Code editor: enhanced [Find and replace features](#).
- New searchable "property" Language element in the [Find in design...](#) dialog box.
- To simplify code, some comparison operators can now be used with [Undefined values](#) without generating errors.
- Support of *headerOnly* parameter in [POP3Transporter.getMail\(\)](#).
- Support of `count values` option in `entitySelection.distinct()` and `collection.distinct()` functions.
- New `entitySelection.distinctPaths()` function.
- Support of `count values` option in `entitySelection.distinct()` and `collection.distinct()` functions.
- ORDA requests logs are now available [on the server](#), new parameter for `.startRequestLog()` function.
- New tools for code execution in CLI: `tool4d` and [4D Server in utility mode](#).
- [Data Explorer](#): new button and display in a 4D window.
- New properties for buttons, check boxes and radio buttons: [Image hugs title](#) and [Horizontal Alignment](#).
- Support of `WinIcon` in `file.setAppInfo()` function.
- New `validateTLSCertificate` option for [4D.HTTPRequest.new\(\)](#) allowing you to control the automatic certificate validation.
- 4D Language commands: [What's new page](#) on doc.4d.com.
- 4D Write Pro: [What's new page](#) on doc.4d.com.
- [Fixed bug list](#): list of all bugs that have been fixed in 4D v20.

### Behavior changes

- For HTTP RFC compliance, `HTTPRequestClass.response.headers` property now returns all header names **in lowercase**. If you want your code to continue working as before, use the new `HTTPRequestClass.response.rawHeaders` property.
- TLS certificates are now automatically validated by 4D when sending HTTP requests with `4D.HTTPRequest.new()`, and rejected with an error if they are invalid. A new *option* property allows you to control this validation.
- TLS v1.0 and TLS v1.1 are deprecated, they are no longer supported as `Min TLS version` on 4D Server. Version 1.3 is now selected by default and is automatically used if `_o_TLSv1_0` or `_o_TLSv1_1` constants are set with [SET DATABASE PARAMETER](#).
- For consistency, all buttons, checkboxes, and radio buttons are now rendered with a "3D" type at runtime: respectively `Object type 3D button`, `Object type 3D checkbox`, and `Object type 3D radio button` are returned by [OBJECT Get type](#) for these objects.
- As of 4D v20, [4D for Mobile](#) is no longer installed by default in the 4D environment. To benefit from the 4D for Mobile development features in 4D, you need to [install the 4D Mobile App component](#) in the "[Components](#)" folder of your projects. If a converted project uses features from the [4D Mobile App Server component](#), make sure you also install it in the "Components" folder of the project.

## 4D v19 R8

Read [What's new in 4D v19 R8](#), the blog post that lists all new features and enhancements in 4D v19 R8.

### Highlights

- Error management methods can be installed for [global and component execution contexts](#).
- Listboxes with collection or entity selection datasources now support [Automatic Row Height](#) column property.
- ORDA: Support of the `roles.json` file to define `privileges` assigned to the session with `setPrivileges()`.
- Support of [SDI mode in test application mode](#) on Windows.
- 4D View Pro:
  - support of themes in tables: new [VP SET TABLE THEME](#) and [VP Get table theme](#) commands, support of theme options in [VP CREATE TABLE](#)
  - new [VP Get table dirty rows](#) command
- 4D Language commands: [What's new page](#) on doc.4d.com.
- 4D Write Pro: [What's new page](#) on doc.4d.com.
- [Fixed bug list](#): list of all bugs that have been fixed in 4D v19 R8.

### Behavior changes

- For HTTP RFC compliance, `HTTPRequestClass.response.headers` property now returns all header names **in lowercase**. If you want your code to continue

working as before, use the new `HTTPRequestClass.response.rawHeaders` property.

- When a [bevel button with linked pop-up menu](#) is assigned a standard action, the standard action is no longer generated if a pop-up menu option is selected.
- In Web areas using blink (CEF), dialogs displayed from external scripts are now blocking if not called from a `setTimeout()` JS function. This is due to the current CEF updates, in which dialogs displayed by functions such as `alert()` or `print()` are no longer handled by the OS but by the Web area. See [WA Evaluate JavaScript](#) and [WA EXECUTE JAVASCRIPT](#).

## 4D v19 R7

Read [What's new in 4D v19 R7](#), the blog post that lists all new features and enhancements in 4D v19 R7.

### Highlights

- Related data and computed/alias attributes can be displayed in the [Data Explorer](#).
- New [FileHandle](#) class and new `.open()` function in the [File](#) class.
- [Entity selection Class](#): `.add()` supports an *entitySelection* parameter, `.minus()` supports a *keepOrder* parameter.
- 4D View Pro: new table commands [VP Find table](#), [VP Get table column attributes](#), [VP Get table column index](#), [VP Get tables](#), [VP INSERT TABLE COLUMNS](#), [VP INSERT TABLE ROWS](#), [VP REMOVE TABLE COLUMNS](#), [VP REMOVE TABLE ROWS](#), [VP RESIZE TABLE](#), [VP SET TABLE COLUMN ATTRIBUTES](#).
- Component namespaces are now [displayed in the Explorer](#).
- Text area and Input form objects now support the [corner radius property](#).
- 4D Language commands: [What's new page](#) on doc.4d.com.
- 4D Write Pro: [What's new page](#) on doc.4d.com.
- [Fixed bug list](#): list of all bugs that have been fixed in 4D v19 R7.

### Behavior changes

- For consistency with standard interfaces (e.g. OS file explorers), list box rules for row selection/drag and drop have been modified. A continuous or discontinuous selection of rows can be dragged by simply clicking on and moving a selected row; the **Alt** key is not longer necessary (but can still be used like in previous previous). When the **Shift** or **Ctrl/Command** key is pressed, a mouse click is taken into account when the click is down. For more information on drag and drop in list boxes, you can refer to [this blog post](#) and download [this HDI 4D project](#).
- The 4D internal build numbering has been modified as of 4D v19 R7:
  - releases up to 4D v19 R6 (included) are numbered 282xxx,
  - releases from 4D v19 R7 will be numbered 100xxx.Note that a specific 4D version is still uniquely identified by a branch name and a build number. The build number increases chronologically.
- The ability to use Wakanda/4D Mobile REST protocol to call a project method has been removed. You can use [ORDA data model class functions](#) or [/4DACTION urls](#) instead.

## 4D v19 R6

- New [HTTPRequest](#) class.
- Collection functions that can call code now support function objects as *formula* parameter: `.every()`, `.filter()`, `.find()`, `.findIndex()`, `.map()`, `.orderByMethod()`, `.reduce()`, `.some()`, `.sort()`

- Listbox cells support [horizontal](#) and [vertical](#) padding.
- 4D View Pro: new [VP CREATE TABLE](#) and [VP REMOVE TABLE](#) commands to handle tables in sheets.
- Ability to see related, computed, and alias attributes in the [Web Data Explorer](#).
- To help us make our products always better, we now automatically collect data regarding usage statistics on running 4D Server applications. This will have no impact on performance. See the new page explaining [why and how 4D collects data](#).
- Components compiled for Silicon: On macOS Silicon platforms (Apple ARM CPUs), components must be recompiled with 4D v19 R6 or higher to be used with this release.

## 4D v19 R5

- The project [directory.json file](#) can now be [embedded in the server](#) at build time, allowing you to deploy a client/server application with a basic security user and group configuration.
- You can now [deselect useless modules](#) in your built applications.
- The *MeCab* library is included by default in all 4D applications on macOS. In previous releases, this library, specifically designed to manage Japanese text, was only available in the Japanese version of 4D on macOS. If you do not need this library in your final applications, you can now [deselect it](#).
- [Client/Server optimization](#): New class functions allow you to handle the ORDA cache and the contents of an optimization context. See [Preconfiguring contexts](#) and [ORDA Cache](#) for more information.

These functions are intended for developers who need to customize ORDA default features for specific configurations. In most cases, you will not need to use them.

- [DataClass class](#): The new `.getCount()` function returns the number of entities in a dataclass.
- The *4DDiagnosticLog.txt* file only records high-level information by default ([INFO level](#)). You can now select the information level to record (for example DEBUG level information) using the `Diagnostic log level` selector of the `SET DATABASE PARAMETER` command or the log configuration file.
- Calling `Use()` on a non-shared object or a non-shared collection does nothing (it no longer generates an error). Thus, it is now useless to test if the object or collection passed to `Use()` is actually shared.
- For clarification purposes, two SQL commands have been prefixed: `GET DATA SOURCE LIST` has been renamed to `SQL GET DATA SOURCE LIST`, `Get current data source` has been renamed to `SQL Get current data source`.
- **4D View Pro:**
  - The new [VP SET DATA CONTEXT](#), [VP Get data context](#), [VP SET BINDING PATH](#), [VP Get binding path](#) commands allow you to create data contexts and bind their contents to sheet cells.
  - [VP EXPORT DOCUMENT](#) and [VP Export to object](#) now accept the new `includeBindingSource` option that exports the contents of a data context as cell values.
  - (Windows only) 4D View Pro areas now use a new print settings window.
- **Web areas:**
  - New Windows system rendering engine: Web Areas using the [Windows system rendering engine](#) are now based upon **Microsoft Edge WebView2**. This impacts the following features:
    - The `WA Create URL history menu` and `WA GET URL HISTORY` commands only return the current URL.

- The [Progression variable](#) is no longer updated.
- Drag and drop features are handled by a Windows API which is tagged "experimental" by Microsoft. Consequently, web areas may not work as expected when this API is not installed: Drag and drop may seem allowed even when the `WA enable URL drop` preference has been set to False. However, the drop action is blocked by default, and you can control the allowed URLs using the [On Window Opening Denied event](#) (see below).
- (Windows only) When the user selects **Print...** from a web area using the embedded web rendering engine, a new print settings window is now displayed.
- To reflect their actual effect in web areas (increase or decrease page zoom level), two commands have been renamed: `WA SET PAGE TEXT LARGER` has been renamed `WA ZOOM IN`, `WA SET PAGE TEXT SMALLER` has been renamed `WA ZOOM OUT`.
- Enhanced security in web areas that use the [embedded web rendering engine](#) or the [Windows system rendering engine](#) (based on Microsoft Edge WebView2):
  - CORS policies now apply when accessing files on disk in web areas. For example, when opening a .html file with `WA OPEN URL`, that .html file cannot contain links that point to files outside its folder
  - Dropping external contents in web areas is now always blocked and triggers the [On Window Opening Denied event](#) when the `WA enable URL drop` preference is set to True (when set to False, the `WA enable URL drop` preference only modifies the drop cursor icon and filters the [On Window Opening Denied event](#)). To allow a drop action, you need to execute additional code in the object method of the web area:

```
WA OPEN URL(*;"WebArea";WA Get last filtered URL(*;"WebArea"))
```

## 4D v19 R4

- [Alias attributes](#) are available in ORDA classes.
- Support for [break and continue](#) statements in loops.
- Support for [return](#) statement and [return expression](#) to return values.
- Support for [compound assignment operators](#), [short-circuit operators](#), and [ternary operator](#)
- The [Code Editor](#) now includes an dropdown tool and supports markers for better code navigation.
- New Preferences: [Include tokens in project source files](#) and [Show clipboards](#) option on the Methods page.
- New REST request to [lock/unlock](#) entities.
- [4D View Pro](#) chapter added with new commands: [VP Copy to object](#), [VP MOVE CELLS](#), [VP PASTE FROM OBJECT](#).
- New [SystemWorker class](#).
- The `Alias selection` constant has been renamed `Allow alias files` to resolve a conflict resulting from the support of alias attributes in ORDA.
- For better compliance with ORDA specifications, the *Map NULL values to blank values* field property is now unchecked by default in databases created with 4D v19 R4 and higher. You can also enable this default behavior in your databases converted from previous versions by selecting the Map NULL values to blank values unchecked by default at field creation compatibility setting. Working with Null values is now recommended since they are fully supported by ORDA.
- Because of the support of the [ternary operator](#), the colon ":" is no longer allowed in variable, field, constant, function, plugin and project method names. If your database/project contains identifiers with colons, you must replace them before

converting it to v19 R4 or higher, otherwise errors may occur in your code. For example, if you have a variable named `a:b`, it could be interpreted as ternary operator syntax:

```
$value:=($size>1000)? a:b // Here 'a:b' is viewed as a ternary operator.
```

## 4D v19 R3

- [Computed properties](#) are available in classes.
- [Computed attributes](#) are available in ORDA classes. They are similar to computed properties but also support [query](#) and [orderBy](#) functions.
- New ORDA dataclass attributes: [exposed](#) and [readOnly](#).
- [ZIP archives](#) now supports LZMA and xz compression algorithms.
- A [new build option](#) makes it easier to include Silicon Mac clients in Server applications on Windows.
- Extended [support of dark mode](#) on macOS.
- Support of **OAuth2 token object** in [IMAP New transporter](#), [POP3 New transporter](#), and [SMTP New transporter](#).
- Users can now load a [log configuration file](#) using a button in the [server administration window](#).
- Handling [optional parameters](#) is more flexible in the 4D language when accessing parameters declared but not passed. For example, the following code no longer provokes an error:

```
// "concat" function of myClass
Function concat ($param1 : Text ; $param2 : Text)
ALERT($param1+" "+$param2)
    // Calling method
$class:=cs.myClass.new()
$class.concat("Hello";" world") // Displays "Hello world"
$class.concat("Hello") // Displays "Hello "
$class.concat() // Displays " "
```

For detailed information, please refer to [this blog post](#). To benefit from this overall simplification, you need to recompile both calling and called methods; thus components must be recompiled.

- Debugging web server sessions [is easier on 4D Server](#).
- The new [4D NetKit](#) component allows you to connect to third-party APIs such as Microsoft Graph.
- 4D v19 R3 uses a stronger hashing algorithm for 4D user passwords: Bcrypt. This new algorithm is automatically used when a password is changed using the Tool Box, the [CHANGE PASSWORD](#) command, or the [Set user properties](#) command. Once a password is modified, opening the database with a version prior to 4D v19 R3 will cause an authentication denial for this account. If you use 4D passwords, it is highly recommended to backup the .4db file (binary databases) or directory.json file (projects) before upgrading to 4D v19 R3 or later.
- For accuracy, the [4D digest](#) constant has been renamed [4D REST digest](#).
- End-of-line and BOM management for XML commands: When opened in 4D v19 R3, projects or databases created with previous releases behave differently regarding default end-of-line characters and BOM management in XML documents: line feed (LF) characters are used instead of CR (on macOS), and byte order marks (BOM) are not included. This allows a better compatibility with VCS tools. If necessary, you can restore the v19 R2 behavior using the [XML SET](#)

`OPTIONS` command. In projects or databases converted from versions prior to 19 R2, these options are managed by two compatibility settings.

- Runtime Explorer shortcut removed in built projects: The **Cmd/Ctrl+Shift+F9** shortcut does no longer display the Runtime Explorer window in single-user merged project applications. This shortcut can now be a user application shortcut. You can call the Runtime Explorer window using the new `OPEN RUNTIME EXPLORER` command.
- Extended debugging capabilities with 4D Server: In interpreted mode, 4D Server can now debug all kinds of processes, including scalable web sessions. This is available when the debugger is attached to the server or to a remote client.  
*Warning: In interpreted mode, in order to make extended debugging available on the server machine, all server processes are now automatically executed in cooperative mode when the debugger is attached to the server (default setting). This can have a significant impact on the performance of your converted applications when they run with 4D Server v19 R3 and higher. To restore preemptive execution on the server in this case, all you need to do is detach the debugger from the server (and attach it to a remote client if necessary).*
- On Windows, 4D projects and databases created with 4D v19 R3 and higher use the [DirectWrite API](#) in forms. This API improves text rendering, especially in high DPI configurations. DirectWrite is used for text rendering with static and input text, checkboxes, buttons, and radio buttons. Note that listboxes already use DirectWrite. A compatibility option allows you to enable DirectWrite in projects and databases created with previous 4D versions.
- If you use components compiled with 4D v19.0 for Silicon (Apple ARM CPUs) which call the `Count parameters` command, we recommend to recompile them with 4D v19 R3 to provide compatibility with 4D v19 R3 and future releases. If a component is not compiled for Silicon, there is no need to recompile.

## 4D v19 R2

- A [default .gitignore file](#) can be created with new projects
- New [Blob class API](#) to handle new [4D.Blob objects](#)
- `no-bom` support and new default end-of-line characters in [.setText\(\)](#)

## Previous releases

- Click to see the release notes for previous versions

## Library table

<b>Library</b>	<b>Current version</b>	<b>Updated in 4D</b>	<b>Comment</b>
ICU	72.1	20	This major upgrade forces an automatic rebuild of alphanumeric, text and object indexes.
CEF	109	20	Chromium 5414. CORS policies now also apply when accessing files on disk (see the "security" paragraph on web areas in the <a href="#">4D v19 R5</a> section above).
Hunspell	1.7.2	20	Used for spell checking in 4D forms and 4D Write Pro
PDFWriter4.3		20	FreeType dependency in 12.2.1
SpreadJS	16.0.4	20	4D View Pro engine
OpenSSL	3.1.1	20	
libZip	1.9.2	20	Used by zip class, 4D Write Pro, svg and serverNet components
LZMA	5.4.1	20	
Zlib	1.2.13	20	
webkit	WKWebView 19		
PHP	8.2.4	20	
libldap	2.4.48	18 R2	
libsasl	2.1.28	20	

# Project Management

A 4D project contains all of the source code of a 4D application, whatever its deployment type (web, mobile, or desktop), from the database structure to the user interface, including code, forms, menus, user settings, or any required resources. A 4D project is primarily made of text-based files.

## Project files

4D project files are open and edited using regular 4D platform applications (4D or 4D Server). With 4D, full-featured editors are available to manage files, including a structure editor, a Code Editor, a form editor, a menu editor...

Since projects are in human-readable, plain text files (JSON, XML, etc.), they can be read or edited manually by developers, using any code editor.

In addition, 4D project files make it easier to program generically, create application templates, and share code. Project are organized internally in [folders and files](#).

## Development

4D projects are developed using the **4D** application. It provides an Integrated Development Environment (IDE) for 4D projects as well as a web server, a mobile project generator, and an application runtime, allowing you to develop, test, and debug any kind of project.

Multi-user development is managed via standard **source control** repository tools (Perforce, Git, SVN, etc.), which allow developers to work on different branches, and compare, merge, or revert modifications.

## Final application

Project files can be [compiled](#) and easily deployed. 4D allows you to create three types of applications from your projects:

- [web](#) applications,
- [mobile](#) applications,
- [desktop](#) applications (client/server or single-user).

Back end applications can be deployed using 4D Server, 4D, or [merged with 4D Volume license](#).

# Architecture of a project

A 4D project is made of several folders and files, stored within a project root folder (package folder). For example:

- MyPackage (*project root folder*)
  - Components
  - Data
    - Logs
    - Settings
  - Documentation
  - Plugins
  - Project
    - DerivedData
    - Sources
    - Trash
  - Resources
  - Settings
  - userPreferences.jSmith
  - WebFolder

If your project has been converted from a binary database, additional folders may be present. See "Converting databases to projects" on [doc.4d.com](http://doc.4d.com).

## Project folder

The Project folder typically contains the following hierarchy:

- <applicationName>.4DProject file
- Sources
  - Classes
  - DatabaseMethods
  - Methods
  - Forms
  - TableForms
  - Triggers
- DerivedData
- Trash (if any)

### <applicationName>.4DProject file

Project development file, used to designate and launch the project. This file can be opened by:

- 4D
- 4D Server (read-only, see [Opening a remote project](#))

In 4D projects, development is done with 4D and multi-user development is managed through source control tools. 4D Server can open .4DProject files for testing purposes.

This text file can also contain configuration keys, in particular `"tokenizedText": false`.

## Sources

Contents	Description	Format
catalog.4DCatalog	Table and field definitions	XML
folders.json	Explorer folder definitions	JSON
menus.json	Menu definitions	JSON
settings.4DSettings	<i>Structure</i> database settings. They are not taken into account if <a href="#">user settings</a> or <a href="#">user settings for data</a> are defined (see also <a href="#">Priority of settings</a> ). <b>Warning:</b> In compiled applications, structure settings are stored in the .4dz file (read-only). For deployment needs, it is necessary to <a href="#">enable</a> and use <i>user settings</i> or <i>user settings for data</i> to define custom settings.	XML
tips.json	Defined tips	JSON
lists.json	Defined lists	JSON
filters.json	Defined filters	JSON
styleSheets.css	CSS style sheets	CSS
styleSheets_mac.css	Mac css style sheets (from converted binary database)	CSS
styleSheets_windows.css	Windows css style sheets (from converted binary database)	CSS

## DatabaseMethods

Contents	Description	Format
databaseMethodName.4dm	Database methods defined in the project. One file per database method	text

## Methods

Contents	Description	Format
methodName.4dm	Project methods defined in the project. One file per method	text

## Classes

Contents	Description	Format
className.4dm	User class definition method, allowing to instantiate specific objects. One file per class, the name of the file is the class name	text

## Forms

Contents	Description	Format
formName/form.4DForm	Project form description	json
formName/method.4dm	Project form method	text
formName/Images/pictureName	Project form static picture	picture
formName/ObjectMethods/objectName.4dm	Object methods. One file per object method	text

## TableForms

Contents	Description	Format
<i>n/Input/formName/form.4DForm</i>	Input table form description (n is the table number)	json
<i>n/Input/formName/Images/pictureName</i>	Input table form static pictures	picture
<i>n/Input/formName/method.4dm</i>	Input table form method	text
<i>n/Input/formName/ObjectMethods/objectName.4dm</i>	Input form object methods. One file per object method	text
<i>n/Output/formName/form.4DForm</i>	Output table form description (n is the table number)	json
<i>n/Output/formName/Images/pictureName</i>	Output table form static pictures	picture
<i>n/Output/formName/method.4dm</i>	Output table form method	text
<i>n/Output/formName/ObjectMethods/objectName.4dm</i>	Output form object methods. One file per object method	text

## Triggers

Contents	Description	Format
<i>table_n.4dm</i>	Trigger methods defined in the project. One trigger file per table (n is the table number)	text

**Note:** The .4dm file extension is a text-based file format, containing the code of a 4D method. It is compliant with source control tools.

## Trash

The Trash folder contains methods and forms that were deleted from the project (if any). It can contain the following folders:

- Methods
- Forms
- TableForms

Within these folders, deleted element names are in parentheses, e.g. "(myMethod).4dm". The folder organization is identical to the [Sources](#) folder.

## DerivedData

The DerivedData folder contains cached data used internally by 4D to optimize processing. It is automatically created or recreated when necessary. You can ignore this folder.

## Libraries

This folder is used on macOS only.

The Libraries folder contains the file resulting from a compilation with the [Silicon compiler](#) on macOS.

## Resources

The Resources folder contains any custom project resource files and folders. In this folder, you can place all the files needed for the translation or customization of the application interface (picture files, text files, XLIFF files, etc.). 4D uses automatic mechanisms to work with the contents of this folder, in particular for the handling of XLIFF files and static pictures. For using in remote mode, the Resources folder lets you share files between the server machine and all the client machines. See the *4D Server Reference Manual*.

Contents	Description	Format
<i>item</i>	Project resource files and folders Pictures from the Picture Library as separate files(*).	various
Images/Library/ <i>item</i>	Names of these items become file names. If a duplicate exists, a number is added to the name.	picture

(\*) only if the project was exported from a .4db binary database.

## Data

The data folder contains the data file and all files and folders relating to the data.

Contents	Description	Format
<i>data.4dd</i> (*)	Data file containing data entered in the records and all the data belonging to the records. When you open a 4D project, the application opens the current data file by default. If you change the name or location of this file, the <i>Open data file</i> dialog box will then appear so that you can select the data file to use or create a new one	binary
<i>data.journal</i>	Created only when the database uses a log file. The log file is used to ensure the security of the data between backups. All operations carried out on the data are recorded sequentially in this file. Therefore, each operation on the data causes two simultaneous actions: the first on the data (the statement is executed normally) and the second in the log file (a description of the operation is recorded). The log file is constructed independently, without disturbing or slowing down the user's work. A database can only work with a single log file at a time. The log file records operations such as additions, modifications or deletions of records, transactions, etc. It is generated by default when a database is created.	binary
<i>data.match</i> (internal)	UUID matching table number	XML

(\*) When the project is created from a .4db binary database, the data file is left untouched. Thus, it can be named differently and placed in another location.

## Settings (user data)

This folder contains [user settings for data](#) used for application administration.

These settings take priority over [user settings](#) and [structure settings](#). See also [Priority of settings](#).

Contents	Description	Format
directory.json	Description of 4D groups, users, and their access rights when the application is run with this data file.	JSON
Backup.4DSettings	Database backup settings, used to set the <a href="#">backup options</a> when the database is run with this data file. Keys concerning backup configuration are described in the <i>4D XML Keys Backup</i> manual.	XML
settings.4DSettings	Custom database settings for this data file.	XML

## Logs

The Logs folder contains all log files used by the project. Log files include, in particular:

- database conversion,
- web server requests,
- backup/restore activities journal (*Backup Journal[xxx].txt*, see [Backup journal](#))
- command debugging,
- 4D Server requests (generated on client machines and on the server).

An additional Logs folder is available in the system user preferences folder (active 4D folder, see [Get 4D folder](#) command) for maintenance log files and in cases where data folder is read-only.

## Settings (user)

This folder contains [user settings](#) used for application administration.

These settings take priority over [structure settings](#) file. However, if [user settings for data](#) exist, they take priority over the user settings. See also [Priority of settings](#).

Contents	Description	Format
directory.json	Description of 4D groups and users for the application, as well as their access rights	JSON
Backup.4DSettings	Database backup settings, used to set the <a href="#">backup options</a> when each backup is launched. This file can also be used to read or set additional options, such as the amount of information stored in the <i>backup journal</i> . Keys concerning backup configuration are described in the <i>4D XML Keys Backup</i> manual.	XML
BuildApp.4DSettings	Build settings file, created automatically when using the application builder dialog box or the <a href="#">BUILD APPLICATION</a> command	XML
settings.4DSettings	Custom settings for this project (all data files)	XML
logConfig.json	Custom <a href="#">log configuration file</a>	json

## userPreferences.<userName>

This folder contains files that memorize user configurations, e.g. break point or window positions. You can just ignore this folder. It contains for example:

<b>Contents</b>	<b>Description</b>	<b>Format</b>
methodPreferences.json	Current user Code Editor preferences	JSON
methodWindowPositions.json	Current user window positions for methods	JSON
formWindowPositions.json	Current user window positions for forms	JSON
workspace.json	List of opened windows; on macOS, order of tab windows	JSON
debuggerCatches.json	Caught calls to commands	JSON
recentTables.json	Ordered list of tables	JSON
preferences.4DPreferences	Current data path and main window positions	XML
CompilerIntermediateFiles	Intermediate files resulting from Apple Silicon compilation	Folder

## Components

This folder contains the components to be available in the application project. It must be stored at the same level as the Project folder.

An application project can be used itself as a component:

- for development: put an alias of the .4dproject file in the Components folder of the host project.
- for deployment: [build the component](#) and put the resulting .4dz file in a .4dbase folder in the Components folder of the host application.

## Plugins

This folder contains the plug-ins to be available in the application project. It must be stored at the same level as the Project folder.

## Documentation

This folder contains all documentation files (.md) created for the project elements such as classes, methods, or forms. Documentation files are managed and displayed in the 4D Explorer.

For more information, refer to [Documenting a project](#).

## WebFolder

Default root folder of the 4D Web server for pages, pictures, etc. It is automatically created when the Web server is launched for the first time.

## .gitignore file (optional)

File that specifies which files will be ignored by git. You can include a gitignore file in your projects using the **Create .gitignore file** option on the **General** page of the preferences. To configure the contents of that file, see [Create .gitignore file](#).

# Documenting a project

In application projects, you can document your methods as well as your classes, forms, tables, or fields. Creating documentation is particularly appropriate for projects being developed by multiple programmers and is generally good programming practice. Documentation can contain a description of an element as well as any information necessary to understand how the element functions in the application.

The following project elements accept documentation:

- Methods (database methods, component methods, project methods, form methods, 4D Mobile methods, and triggers)
- Classes
- Forms
- Tables and Fields

Your documentation files are written in Markdown syntax (.md files) using any editor that supports Markdown. They are stored as independant files within your project folder.

Documentation is displayed in the preview area (right-side panel) of the Explorer:

It can also be partially exposed as [code editor tips](#).

## Documentation files

### Documentation file name

Documentation files have the same name as their attached element, with the ".md" extension. For example, the documentation file attached to the `myMethod.4dm` project method will be named `myMethod.md`.

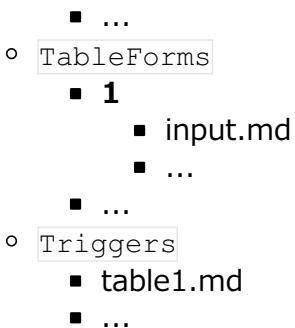
In the Explorer, 4D automatically displays the documentation file with the same name as the selected element (see below).

### Documentation file architecture

All documentation files are stored in the `Documentation` folder, located at the first level of the package folder.

The `Documentation` folder architecture is the following:

- `Documentation`
  - `Classes`
    - `myClass.md`
  - `DatabaseMethods`
    - `onStartup.md`
    - ...
  - `Forms`
    - `loginDial.md`
    - ...
  - `Methods`
    - `myMethod.md`



- A project form and its project form method share the same documentation file for form and method.
- A table form and its table form method share the same documentation file for form and method.

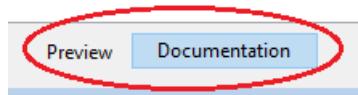
Renaming or deleting a documented element in your project will also rename or delete the element's associated Markdown file.

## Documentation in the Explorer

### Viewing documentation

To view documentation in the Explorer window:

1. Make sure the preview area is displayed.
2. Select the documented element in the Explorer list.
3. Click the **Documentation** button located below the preview area.



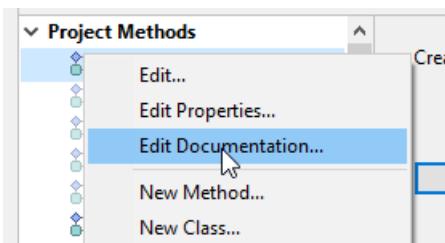
- If no documentation file was found for the selected element, a **Create** button is displayed (see below).
- Otherwise, if a documentation file exists for the selected element, the contents are displayed in the area. The contents are not directly editable in the pane.

### Editing documentation file

You can create and/or edit a Markdown documentation file from the Explorer window for the selected element.

If there is no documentation file for the selected element, you can:

- click on the **Create** button in the **Documentation** pane or,
- choose the **Edit Documentation...** option in the contextual menu or options menu of the Explorer.



4D automatically creates an appropriately named .md file with a basic template at the

relevant location and opens it with your default Markdown editor.

If a documentation file already exists for the selected element, you can open it with your Markdown editor by choosing the **Edit Documentation...** option in the contextual menu or options menu of the Explorer.

## Viewing documentation in the code editor

The 4D code editor displays a part of a method's documentation in its help tip.

If a file named `\<MethodName>.md` exists in the `\<package>/documentation` folder, the code editor displays (by priority):

- Any text entered in an HTML comment tag (`<!-- command documentation -->`) at the top of the markdown file.
- Or, if no html comment tag is used, the first sentence after a `# Description` tag of the markdown file.  
In this case, the first line contains the **prototype** of the method, automatically generated by the 4D code parser.

### NOTE

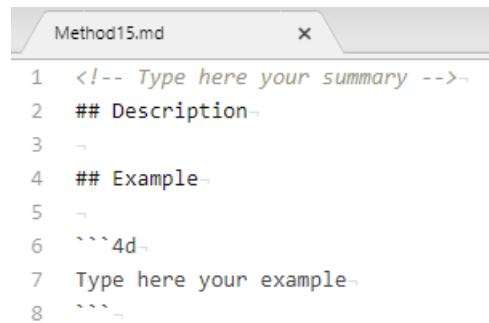
Otherwise, the code editor displays [the block comment at the top of the method code](#).

## Documentation file definition

4D uses a basic template to create new documentation files. This template suggests specific features that allow you to [display information in the code editor](#).

However, you can use any [supported Markdown tags](#).

New documentation files are created with the following default contents:



```
Method15.md
1 <!-- Type here your summary -->
2 ## Description
3
4 ## Example
5
6 ````4d
7 Type here your example
8 ````
```

Line	Description
<code>&lt;!-- Type here your summary --&gt;</code>	HTML comment. Used in priority as the method description in the <a href="#">code editor tips</a>
<code>## Description</code>	Heading level 2 in Markdown. The first sentence after this tag is used as the method description in the code editor tips if HTML comment is not used
<code>## Example</code>	Heading level 2, you can use this area to show sample code
<code>````4d Type here your example````</code>	Used to format 4D code examples (uses highlight.js library)

## Supported Markdown

- The title tag is supported:

```
# Title 1  
## Title 2  
### Title 3
```

- The style tags (*italic*, **bold**, ~~strikethrough~~) are supported:

```
_italic_  
**bold**  
**_bold/italic_*  
~~strikethrough~~
```

- The code block tag (```4d ... ```) is supported with 4D code highlight:

```
```4d  
var $txt : Text  
$txt:="Hello world!"  
```
```

- The table tag is supported:

| Parameter | Type   | Description    |
|-----------|--------|----------------|
| wpArea    | String | Write pro area |
| toolbar   | String | Toolbar name   |

- The link tag is supported:

```
// Case 1  
The [documentation] (https://doc.4d.com) of the command ....  
  
// Case 2  
[4D blog][1]  
  
[1]: https://blog.4d.com
```

- The image tags are supported:

```
![image info] (pictures/image.png)  
  
![logo 4D] (https://blog.4d.com/wp-content/uploads/2016/09/logoOriginal-1.png "4D blog logo")  
  
[![logo 4D blog with link] (https://blog.4d.com/wp-content/uploads/2016/09/logoOriginal-1.png "4D blog logo") ]  
(https://blog.4d.com)
```



For more information, see the [GitHub Markdown guide](#).

## Example

In the `WP_SwitchToolbar.md` file, you can write:

```
<!-- This method returns a different logo depending on the size parameter -->
```

```
GetLogo (size) -> logo
```

| Parameter | Type    | in/out | Description                  |
|-----------|---------|--------|------------------------------|
| size      | Longint | in     | Logo style selector (1 to 5) |
| logo      | Picture | out    | Selected logo                |

```
## Description
```

This method returns a logo of a specific size, depending on the value of the \*size\* parameter.

1 = smallest size, 5 = largest size.

```
## Example
```

```
```4d
C_PICTURE($logo)
C_LONGINT($size)

//Get the largest logo
$logo:=GetLogo(5)
```

```

- Explorer view:

- Code editor view:

### **GetLogo**

```
GetLogo (Longint) -> Picture
This method returns a different logo depending on the size parameter
```

# Compilation

You can compile your projects, i.e., translate all of your methods into machine language. Compiling a project lets you check the consistency of the code and accelerate its execution, as well as making it possible to obfuscate the code in its entirety. Compilation is an indispensable step between the development of projects using 4D and their deployment as stand-alone applications.

## Compile

The compilation is handled from your 4D application and is entirely automatic.

On macOS, the compilation requires that you install `xcode`. See [this section](#) for more information about this requirement.

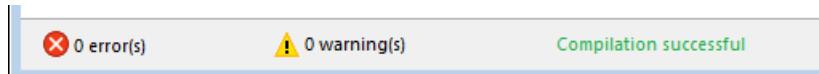
1. Open the compiler window by selecting the **Compiler...** command in the **Design** menu or the **Compiler** toolbar button.



You can also launch directly the compilation by selecting the **Start Compilation** menu item from the **Design** menu.

2. Click the **Compile** button to launch the compilation using the current [compilation settings](#).

If no errors are detected, the actual compilation begins and the "Compilation successful" message is displayed at the bottom of the window when the compilation is completed:



You can immediately [run your application in compiled mode](#) and see how faster it is.

If errors are detected, the process is stopped and the "Compilation failed" message is displayed. The information area of the window displays the method names and line numbers concerned in a hierarchical list:

Double-click on each error detected to open the method or class concerned directly in the 4D Code Editor. The line containing the error is highlighted and the type of error is displayed in the syntax area of the window.

Use the **Previous Error / Next Error** commands of the **Method** menu to navigate from one error to the next.

The number of errors found during your first compilations may be daunting, but do not let this put you off. You will soon discover that they often spring from the same source, i.e., non-compliance with certain project conventions. The compiler always provides a [precise diagnosis](#) of the errors in order to help you correct them.

Compilation requires an appropriate license. Without this license, it is not possible to carry out a compilation (buttons are disabled). Nevertheless, it is still possible to check the syntax and generate Typing methods.

## Run Compiled

Once a project is compiled, it is possible to switch from [interpreted mode to compiled mode](#), and vice versa, at any time and without having to quit the 4D application (except when the interpreted code has been removed). To do this, use the **Restart Interpreted** and **Restart Compiled** commands of the **Run** menu. The [Open project dialog box](#) also offers a choice between interpreted or compiled mode for database startup.

When you switch from one mode to the other, 4D closes the current mode and opens the new one. This is equivalent to exiting and reopening the application. Each time you change from one mode to another, 4D executes the two following database methods (if specified) in this order: `On Exit` -> `On Startup`.

If you modify your project in interpreted mode, you must recompile it in order to have your edits taken into account in compiled mode.

## Compiler window features

In addition to the [Compile button](#), the Compiler window provides additional features that are useful during the project development phase.

### Check Syntax

The **Check Syntax** button starts the execution of the syntax-checking phase. At the end of the checking process, any errors detected are listed in the information area. You can double-click on an error line in order to display the corresponding method.

Syntax checking can also be launched directly using the **Check Syntax** command associated with the **Compiler** toolbar button. This option is the only one available if you do not have a suitable license to allow the compilation of applications.

### Generate Typing

The **Generate Typing** button creates or updates typing compiler methods. Compiler methods are project methods that group together all the variable and array typing declarations (process and interprocess), as well as the method parameters. These methods, when they exist, are used directly by the compiler during code compilation, resulting in faster compilation times.

The name of these methods must begin with `Compiler_`. You can set the default name for each of the 5 compiler methods in the [compiler settings window](#). The compiler methods that are generated and maintained by 4D automatically have the `Invisible` attribute:

- ❖ `Compiler_Arrays`
- ❖ `Compiler_Methods`
- ❖ `Compiler_Variables`

Only the necessary compiler methods (i.e., those for which items already exist in the project) are generated.

The information area indicates any errors found during method creation or updating. Double-clicking on an error line causes the method and line concerned to be displayed in the Code Editor.

## Clear compiled code

The **Clear compiled code** button deletes the compiled code of the project. When you click on it, all of the [code generated during compilation](#) is deleted, the **Restart Compiled** command of the **Run** menu is disabled and the "Compiled Project" option is not available at startup.

## Show/Hide Warnings

Warnings are specific messages generated by the compiler when it checks the syntax. These messages are intended to draw your attention to statements that might lead to execution errors. They do not prevent compilation.

Depending on circumstances and the programming style used, these warnings may be more or less relevant. You can toggle the warnings on or off by clicking the **Show/Hide Warnings** button:



When this option is checked, the warnings (if any) are displayed in the window, after the other error types. They appear in italics:

*Manager*  
1: The name of class 'Manager' looks like a data model class but doesn't extend the appropriate superclass.

Double-clicking a warning opens the corresponding method.

## Disabling warnings during compilation

You can selectively disable certain warnings during compilation by inserting the following into the code of a 4D method:

```
//%W-<warning number>
```

Only warnings with numbers can be disabled. Warning numbers are specified at the end of each message in the list of compilation errors. For example, to disable the following warning:

1: Pointer in an array declaration (518.5)

... you just need to write the following comment in a 4D method, preferably a `COMPILER_xxx` method (method compiled first):

```
//%W-518.5
```

## Compiler Settings

The "Compiler" page of the Settings dialog box lets you set parameters related to project compilation. You can directly open this page from the [compiler window](#) by clicking on the **Compiler Settings** button:



## Compilation options

This area groups the generic options used during the compilation process.

## Generate the symbol file

Used to generate the symbol file (see [symbol file](#)). The symbol file is created in the in the [Logs folder](#) of the project with the name `ProjectName_symbols.txt`.

## Generate error file

Used to generate the error file (see [error file](#)) at the time of syntax checking. The error file is created in the [Logs folder](#) of the project with the name `ProjectName_errors.xml`.

## Compilation Path

Used to set the number of passes (code parsing) performed by the compiler and thus the duration of compilation.

- **Type the variables:** Passes by all the stages that make compilation possible.
- **Process and interprocess variables are typed:** The pass for typing process and interprocess variables is not carried out. This option can be used when you have already carried out the typing of all your process and interprocess variables either yourself or using the function for automatic generation of compiler methods.
- **All variables are typed:** The pass for typing local, process and interprocess variables is not carried out. Use this option when you are certain that all the process, interprocess and local variables have been clearly typed.

## Compilation Target

### ▪ History

This setting allows you to select the processor family for which your 4D project must be natively compiled. The 4D compiler can build native code for two processor families:

- **Intel/AMD** processors (all machines),
- **Apple Silicon** processors.

Two target options are proposed. The result depends on the processor of the machine on which 4D is running.

| <i>Option</i>   | <i>on Windows Intel/AMD</i>  | <i>on macOS Intel</i>  | <i>on macOS Silicon</i>  |
|---|--|--|--|
| <b>All processors<br/>(Intel/AMD and<br/>Apple Silicon)</b> | Code for Intel/AMD<br><i>It is not possible to<br/>produce Apple Silicon<br/>code on Windows</i> | Code for Apple<br>Silicon + Code for<br>Intel/AMD<br><i>Two compiled<br/>codes will be<br/>available</i> | Code for Apple<br>Silicon + Code for<br>Intel/AMD<br><i>Two compiled<br/>codes will be<br/>available</i> |
| <b>My processor<br/>(processor)</b>                         | Code for Intel/AMD   | Code for Intel/AMD   | Code for Apple<br>Silicon  |

Apple Silicon compiler target requires that the **Clang** application be installed on your machine. Clang comes with the latest version of Xcode. See the [Silicon compiler requirements](#) for more information.

## Default typing

Use this area to set the default type for ambiguous database objects.

- **Numeric:** Used to force numeric typing in an unambiguous manner, either in real or longint. This will not override the directives you may have set in your project. You can optimize the running of your database by choosing the Longint type.

- **Button:** Used to force button typing in an unambiguous manner, either in real or longint. This will not override the directives you may have set in your project. This type applies to buttons as well as check boxes, picture buttons, button grids, radio buttons, picture pop-up menus and drop-down lists.

## Compiler Methods for...

This area lets you rename the Compiler methods that are generated automatically by the compiler when you click [Generate Typing](#).

Up to 5 compiler methods may be generated; a compiler method is only generated if the project contains the following items:

- **Variables:** Groups together process variable declarations;
- **Interprocess Variables:** Groups together interprocess variable declarations;
- **Arrays:** Groups together process array declarations;
- **Interprocess Arrays:** Groups together interprocess array declarations;
- **Methods:** Groups together method parameter declarations (for instance, `C_LONGINT (mymethod;$a;$b)`). For more information, see [Compiler method](#)

You can rename each of these methods in the corresponding areas, but they will always be preceded by the label `Compiler_` (non-modifiable). The name of each method (prefix included) must be no longer than 31 characters. It must also be unique and comply with [4D rules for naming methods](#).

## Compilation tools

### Symbol file

If you check the [Generate the symbol file](#) option in the compiler settings, a symbol file called `ProjectName_symbols.txt` is created in the [Logs folder](#) of the project during compilation. It is divided into several parts:

### List of process and interprocess variables

These two lists contain four columns:

- Names of process and interprocess variables and arrays used in your project. These variables are listed in alphabetical order.
- Type of the variable. Types are set by compiler directive commands or are determined by the compiler based on the use of the variable. If the type of a variable cannot be determined, the column is empty.
- Number of dimensions if the variable is an array.
- Reference to the context in which the compiler established the type of the variable. If the variable is used in several contexts, the context mentioned is the one used by the compiler to determine its type.

- If the variable was found in a database method, the database method name is given, preceded by (M)\*.
- If the variable was found in a project method, the method is identified as it has been defined in 4D, preceded by (M).
- If the variable was found in a trigger, the table name is given, preceded by (TM).
- If the variable was found in a form method, the form name is given, preceded by the table name and (FM).
- If the variable was found in an object method, the object method's name is given, preceded by the form name, table name, and by (OM).
- If the variable is an object in a form and does not appear in any project, form, object method, or trigger, the name of the form in which it appears is given, preceded by (F).

**At the end of each list, you can find the sizes of the process and interprocess variables in bytes.**

When compiling, the compiler cannot determine in which process a given process variable is used. A process variable can have a different value in each process. Consequently, all process variables are systematically duplicated as each new process is launched: it is thus advisable to watch out for the amount of memory that they will take up. Also, keep in mind that the space for process variables is not related to the stack size for the process.

## **List of local variables**

The list of local variables is sorted by database method, project method, trigger, form method, and object method, in the same order as in 4D.

This list is divided into three columns:

- list of local variables used in the method;
- type of the variable;
- number of dimensions if the variable is an array.

## **Complete list of methods**

A complete list of your database and project methods is given at the end of the file with:

- their type (procedure or function returning a value)
- the data types of their parameters and the returned result
- the number of calls
- the Thread Safe or Thread Unsafe property.

This information appears as follows:

```
Procedure or Function <Method name>(parameter data types) :
result data type, number of calls, Thread Safe or Thread Unsafe
```

## **Error file**

You can choose whether or not to generate an error file during compilation using the [Generate error file](#) option in the compiler settings. The error file is automatically named `projectName_errors.xml` and is placed in the [Logs folder](#) of the project.

Although the errors can be accessed directly via the [compiler window](#), it can be useful to have an error file that can be transmitted from one machine to another. The error file is generated in XML format in order to facilitate automatic parsing of its contents. It also allows the creation of customized error display interfaces.

The length of the error file depends on the number of errors and warnings issued by the compiler.

The structure of the error file is as follows:

- At the top of the file is the list of errors and warnings, sorted by method and in their order of creation in 4D.
- In the **General errors** section, all the typing impossibilities and identity ambiguities are grouped together. These errors and warnings are listed using the following format:
  - line number in the method (0 indicates general errors)
  - warning attribute indicating whether the detected anomaly is a warning (warning="true") or an error (warning="false")
  - diagnostic describing the error

If your project does not have any general errors, the file will not have a *General errors* section.

An error file may contain three types of messages:

- **Errors linked to a specific line:** these errors are displayed in context — the line in which they were found — with an explanation. The compiler reports this type of error when it encounters an expression in which it sees an inconsistency related to data type or syntax. In the compiler window, double-click on each error detected in order to open the method concerned directly in the 4D Code Editor, with the line containing the error highlighted.
- **General errors:** These are errors that make it impossible to compile the project. There are two cases in which the compiler reports a general error:
  - The data type of a process variable could not be determined.
  - Two different kinds of objects have the same name.

General errors are so named because they cannot be linked to any specific method. In the first case, the compiler could not perform a specified typing anywhere in the project. In the second, it was unable to decide whether to associate a given name with one object rather than with another.

- **Warnings:** Warnings are not errors. They do not prevent the project from being compiled, but simply point out potential code errors. In the compiler window, warnings appear in italics. Double-click on each warning to open the method concerned directly in the 4D Code Editor, with the line containing the warning highlighted.

## Range checking

The code generated by the 4D compiler automatically checks that every access to an array element or a character reference is done within the actual range of array elements or string characters. Out of range accesses will provoke runtime execution errors.

In some cases, you might prefer range checking not to apply to certain parts of the code that are considered to be reliable. More particularly, in the case of loops that are repeated a great number of times, and when running the compiled database on older machines, range checking can significantly slow down processing. If you are absolutely certain that the code concerned is reliable and cannot cause system errors, you can disable range checking locally.

To do this, you must surround the code to be excluded from range checking with the

special comments `//%R-` and `//%R+`. The `//%R-` comment disables range checking and `//%R+` enables it again:

```
// %R-    to disable range checking
...
... //Place the code to be excluded from range checking here
// %R+    to enable range checking again for the rest
```

## About Compilers

4D contains two compilers:

- a "classic" compiler, used to compile native code for Intel/AMD processors;
- a Silicon compiler, used to compile native code for Apple Silicon processors.

The classic compiler can be used on any platform, while the Silicon compiler can only be used on a Mac machine:

|                | Compile for Windows | Compile for Intel Mac | Compile for Silicon Mac |
|----------------|---------------------|-----------------------|-------------------------|
| On Windows     | ✓                   | ✓                     | ✗                       |
| On Intel Mac   | ✓                   | ✓                     | ✓                       |
| On Silicon Mac | ✓                   | ✓                     | ✓                       |

Both compilers are integrated into 4D. The appropriate compiler is automatically selected depending on the [compilation target](#) option.

### Classic Compiler

The classic compiler generates native compiled code for Intel/AMD processors on any machines. It does not require any specific configuration.

Resulting compiled code is stored in the [DerivedData](#) folder of the project.

### Silicon Compiler

The Silicon compiler generates native compiled code for Apple Silicon processors, such as *Apple M1*.

Resulting compiled code is stored in the [Libraries](#) folder of the project.

## Requirements

- **Apple machine:** The Silicon compiler can only be run from an Apple machine.
- **4D Project architecture:** The Silicon compiler is only available for 4D developments using [project architecture](#).
- **Xcode or Developer Tools:** The Silicon compiler calls the **Clang** open-source macOS compiler to compile the project from C++ code at the [second step](#) of compilation. *clang* requires Apple native libraries, which are provided by either the **Xcode** or **Developer Tools** package.
  - **If you already have** Xcode or Developer Tools installed on your computer, you only need to make sure that its version is compliant with 4D requirements.
  - **If you do not have** any of these tools installed on your computer, you will need to download one of them from the Apple Developer web site.

We recommend to install **Xcode**, which is quite simple to install. You can decide to install **Developer Tools** which is more compact, however its

installation is a little more complex.

In any cases, the 4D Silicon compiler will warn you if your configuration does not comply with its requirements.

## Incremental compiler

The Silicon compiler is incremental, which means that:

- During the very first compilation, **all 4D methods** are compiled. This step could take a certain time. However it only occurs once.
- During all subsequent compilations, only **new or modified methods** are processed, thus reducing drastically the compilation time.

## About the 4D Language

The 4D built-in language, consisting of more than 1300 commands, makes 4D a powerful development tool for web, mobile, or desktop applications. You can use the 4D language for many different tasks—from performing simple calculations to creating complex custom user interfaces. For example, you can:

- Programmatically access any of the record management editors (order by, query, and so on),
- Create and print complex reports and labels with the information from the database,
- Communicate with other devices,
- Send emails,
- Manage documents and web pages,
- Import and export data between 4D applications and other applications,
- Incorporate procedures written in other languages into the 4D programming language.

The flexibility and power of the 4D programming language make it the ideal tool for all levels of users and developers to accomplish a complete range of information management tasks. Novice users can quickly perform calculations. Experienced users without programming experience can customize their applications. Experienced developers can use this powerful programming language to add sophisticated features and capabilities to their applications, including file transfer, communications, monitoring. Developers with programming experience in other languages can add their own commands to the 4D language.

## What is a Language?

The 4D language is not very different from the spoken language we use every day. It is a form of communication used to express ideas, inform, and instruct. Like a spoken language, 4D has its own vocabulary, grammar, and syntax; you use it to tell 4D how to manage your application and data.

You do not need to know everything in the language in order to work effectively with 4D. In order to speak, you do not need to know the entire English language; in fact, you can have a small vocabulary and still be quite eloquent. The 4D language is much the same—you only need to know a small part of the language to become productive, and you can learn the rest as the need arises.

## Why use a Language?

At first it may seem that there is little need for a programming language in 4D. In the Design environment, 4D provides flexible tools that require no programming to perform a wide variety of data management tasks. Fundamental tasks, such as data entry, queries, sorting, and reporting are handled with ease. In fact, many extra capabilities are available, such as data validation, data entry aids, graphing, and label generation.

Then why do we need a 4D language? Here are some of its uses:

- Automate repetitive tasks: These tasks include data modification, generation of complex reports, and unattended completion of long series of operations.
- Control the user interface: You can manage windows and menus, and control forms and interface objects.

- Perform sophisticated data management: These tasks include transaction processing, complex data validation, multi-user management, sets, and named selection operations.
- Control the computer: You can control serial port communications, document management, and error management.
- Create applications: You can create easy-to-use, customized applications that run stand-alone.
- Add functionality to the built-in 4D Web server: build and update dynamic web pages filled with your data.

The language lets you take complete control over the design and operation of your application. 4D provides powerful “generic” editors, but the language lets you customize your application to whatever degree you require.

## Taking Control of Your Data

The 4D language lets you take complete control of your data in a powerful and elegant manner. The language is easy enough for a beginner, and sophisticated enough for an experienced application developer. It provides smooth transitions from built-in database functions to a completely customized application.

The commands in the 4D language provide access to the standard record management editors. For example, when you use the `QUERY` command, you are presented with the Query Editor (which can be accessed in the Design mode using the Query command in the Records menu. You can tell the command to search for explicitly described data. For example, `QUERY ([People]; [People] Last Name="Smith")` will find all the people named Smith in your database.

The 4D language is very powerful—one command often replaces hundreds or even thousands of lines of code written in traditional computer languages. Surprisingly enough, with this power comes simplicity—commands have plain English names. For example, to perform a query, you use the `QUERY` command; to add a new record, you use the `ADD RECORD` command.

The language is designed for you to easily accomplish almost any task. Adding a record, sorting records, searching for data, and similar operations are specified with simple and direct commands. But the language can also control the serial ports, read disk documents, control sophisticated transaction processing, and much more.

The 4D language accomplishes even the most sophisticated tasks with relative simplicity. Performing these tasks without using the language would be unimaginable for many. Even with the language’s powerful commands, some tasks can be complex and difficult. A tool by itself does not make a task possible; the task itself may be challenging and the tool can only ease the process. For example, a word processor makes writing a book faster and easier, but it will not write the book for you. Using the 4D language will make the process of managing your data easier and will allow you to approach complicated tasks with confidence.

## Is it a “Traditional” Computer Language?

If you are familiar with traditional computer languages, this section may be of interest. If not, you may want to skip it.

The 4D language is not a traditional computer language. It is one of the most innovative and flexible languages available on a computer today. It is designed to work the way you do, and not the other way around.

To use traditional languages, you must do extensive planning. In fact, planning is one

of the major steps in development. 4D allows you to start using the language at any time and in any part of your project. You may start by adding a method to a form, then later add a few more methods. As your application becomes more sophisticated, you might add a project method controlled by a menu. You can use as little or as much of the language as you want. It is not “all or nothing,” as is the case with many other databases.

Traditional languages force you to define and pre-declare interface objects in formal syntactic terms. In 4D, you simply create an object, such as a button, and use it. 4D automatically manages the object for you. For example, to use a button, you draw it on a form and name it. When the user clicks the button, the language automatically notifies your methods.

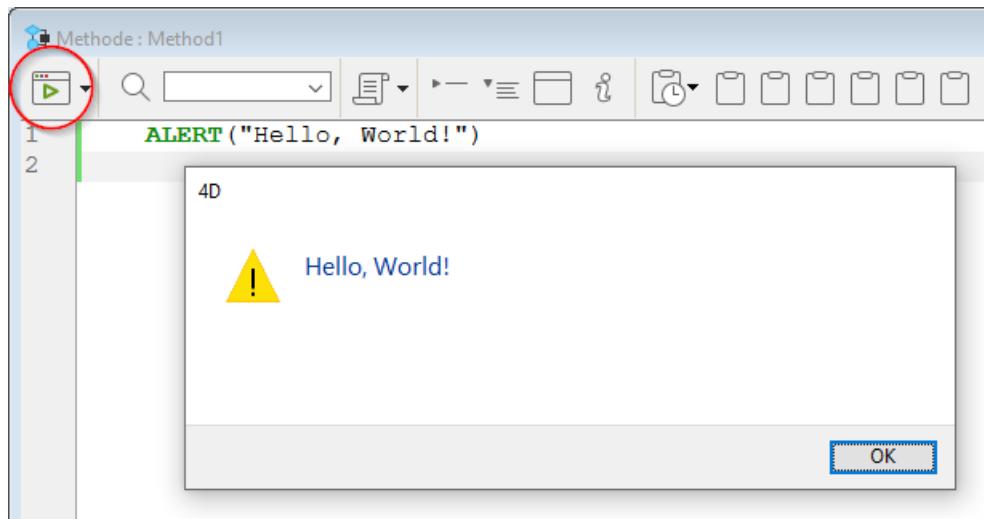
Traditional languages are often rigid and inflexible, requiring commands to be entered in a very formal and restrictive style. The 4D language breaks with tradition, and the benefits are yours.

## A Quick Tour

Using the 4D language, printing the traditional "Hello, world!" message on screen can be done in several ways. The most simple is probably to write the following single line in a project method:

```
ALERT("Hello, World!")
```

This code will display a platform-standard alert dialog box with the "Hello, World!" message, containing an OK button. To execute the code, you just need to click on the execution button in the Code Editor:



Or, you could attach this code to a button in a form and execute the form, in which case clicking on the button would display the alert dialog box. In any cases, you have just executed your first line of 4D code!

## Assigning Values

Data can be put into and copied out of variables, fields, array elements... Putting data into a variable is called assigning the data to the variable and is done with the assignment operator (:=). The assignment operator is also used to assign data to fields or array elements.

```
$MyNumber:=3 //assigns 3 to MyNumber variable  
[Products]Size:=$MyNumber //assigns MyNumber variable to [Products]Size  
field  
arrDays{2}:="Tuesday" //assigns "Tuesday" string to the 2nd arrDays  
element  
MyVar:=Length("Acme") //assigns the result of the function (4) to MyVar  
$myDate:=!2018/01/21! //assigns a date literal  
$myHour:=?08:12:55? //assigns a time literal
```

You MUST distinguish the assignment operator := from the other operators. Rather than combining expressions into a new one, the assignment operator copies the value of the expression to the right of the assignment operator into the variable or field to the left of the operator.

**Important:** Do NOT confuse the assignment operator := with the equality comparison operator =. A different assignment operator (and not =) was deliberately chosen to avoid issues and confusion which often occur with == or === in other programming languages. Such errors are often difficult to recognize by the compiler and lead to

time-consuming troubleshooting.

## Variables

The 4D language is strongly typed, although some flexibility is allowed in many cases. You create a typed variable using the `var` keyword. For example, to create a variable of the date type, you can write:

```
var MyDate : Date
```

The `var` keyword allows declaring object variables of a defined class type, for example:

```
var myPerson : cs.Person  
//variable of the Person user class
```

Even if it is usually not recommended, you can declare variables simply by using them; you do not necessarily need to formally define them. For example, if you want a variable that will hold the current date plus 30 days, you can write:

```
MyOtherDate:=Current date+30
```

The line of code reads "MyOtherDate gets the current date plus 30 days." This line declares the variable, assigns it with both the (temporary) date type and a content. A variable declared by assignment is interpreted as typeless, that is, it can be assigned with other types in other lines and then changes the type dynamically. A variable typed with `var` cannot change the type. In [compiled mode](#) however, the type can never be changed, regardless of how the variable was declared.

## Commands

4D commands are built-in methods to perform an action. Commands are often used with parameters, which are passed in brackets () and separated by semicolons (;). Example:

```
COPY DOCUMENT ("folder1\\name1";"folder2\\\" ; "new")
```

Some commands are attached to collections or objects, in which case they are named functions and are used using the dot notation. For example:

```
$c:=New collection(1;2;3;4;5)  
$nc:=$c.slice(0;3) // $nc=[1,2,3]
```

```
$lastEmployee:=$employee.last()
```

You can use 4D plug-ins or 4D components that add new commands to your 4D development environment.

There are many plug-ins proposed by the 4D user community or 3rd-party developers on the market. For example, using the [4d-plugin-pdf-pages](#) on macOS:

```
PDF REMOVE PAGE (path;page)
```

4D SVG is an example of a utility component extending the capabilities of your application:

```
//drawing a picture  
svgRef:=SVG_New  
objectRef:=SVG_New_arc(svgRef;100;100;90;90;180)
```

4D SVG is included in 4D.

## Constants

4D proposes an extended set of predefined constants, whose values are accessible by name. They allow writing more readable code. For example, `Read Mode` is a constant (value 2).

```
vRef:=Open document("PassFile";"TEXT";Read Mode) // open doc in read only mode
```

Predefined constants appear underlined by default in the 4D Code Editor.

## Methods

4D provides a large number of built-in methods (or commands) but also lets you can create your own **project methods**. Project methods are user-defined methods that contain commands, operators, and other parts of the language. Project methods are generic methods, but there are other kinds of methods: Object methods, Form methods, Table methods (Triggers), and Database methods.

A method is composed of statements; each statement consists of one line in the method. A statement performs an action, and may be simple or complex.

For example, the following line is a statement that will display a confirmation dialog box:

```
CONFIRM("Do you really want to close this account?";"Yes";"No")
```

A method also contains tests and loops that control the flow of the execution. 4D methods support `If...Else...End if` and `Case of...Else...End case` branching structures as well as looping structures: `While...End while`, `Repeat...Until`, `For...End for`, and `For each...End for each`:

The following example goes through all the characters of the text `vtSomeText`:

```
For ($v1Char;1;Length(vtSomeText))
    //Do something with the character if it is a TAB

    If(Character code(vtSomeText[[ $v1Char ]])=Tab)
        //...
    End if
End for
```

A project method can call another project method with or without parameters (arguments). The parameters are passed to the method in parentheses, following the name of the method. Each parameter is separated from the next by a semicolon (;). The parameters are directly available within the called method if they have been declared. A method can return a single value in a parameter, which have to be declared. When you call a method, you just type its name:

```
$myText:="hello"
$myText:=Do_Something($myText) //Call the Do_Something method
ALERT($myText) //"HELLO"

//Here the code of the method Do_Something
#DECLARE ($in : Text) -> $out : Text
$out:=Uppercase($in)
```

## Data Types

In the language, the various types of data that can be handled are referred to as data

types. There are basic data types (string, numeric, date, time, Boolean, picture, pointers, arrays), and also composite data types (BLOBs, objects, collections).

Note that string and numeric data types can be associated with more than one type of field. When data is put into a field, the language automatically converts the data to the correct type for the field. For example, if an integer field is used, its data is automatically treated as numeric. In other words, you need not worry about mixing similar field types when using the language; it will manage them for you.

However, when using the language it is important that you do not mix different data types. In the same way that it makes no sense to store "ABC" in a Date field, it makes no sense to put "ABC" in a variable used for dates. In most cases, 4D is very tolerant and will try to make sense of what you are doing. For example, if you add a number to a date, 4D will assume that you want to add that number of days to the date, but if you try to add a string to a date, 4D will tell you that the operation cannot work.

There are cases in which you need to store data as one type and use it as another type. The language contains a full complement of commands that let you convert from one data type to another. For example, you may need to create a part number that starts with a number and ends with characters such as "abc". In this case, you might write:

```
[Products] Part Number:=String(Number) +"abc"
```

If *Number* is 17, then *[Products]Part Number* will get the string "17abc".

The data types are fully defined in the section [Data Types](#).

## Objects and collections

You can handle 4D language objects and collections using the object notation to get or to set their values. For example:

```
employee.name:="Smith"
```

You can also use a string within square brackets, for example:

```
$vName:=employee ["name"]
```

Since an object property value can be an object or a collection, object notation accepts a sequence of symbols to access sub-properties, for example:

```
$vAge:=employee.children[2].age
```

Note that if the object property value is an object that encapsulates a method (a formula), you need to add parenthesis () to the property name to execute the method:

```
$f:=New object  
$f.message:=Formula(ALERT("Hello world!"))  
$f.message() //displays "Hello world!"
```

To access a collection element, you have to pass the element number embedded in square brackets:

```
var myColl : Collection  
myColl:=New collection("A";"B";1;2;Current time)  
myColl[3] //access to 4th element of the collection
```

## Classes

The 4D language supports object classes. Add a `myClass.4dm` file in the Project/Sources/Classes folder of a project to create a class named "myClass".

To instantiate an object of the class in a method, call the user class from the `class store (cs)` and use the `new()` member function. You can pass parameters.

```
// in a 4D method
$o:=cs.myClass.new()
```

In the `myClass` class method, use the `Function <methodName>` statement to define the `methodName` class member function. A class member function can receive and return parameters like any method, and use `This` as the object instance.

```
//in the myClass.4dm file
Function hello -> $welcome : Text
    $welcome:="Hello "+This.who
```

To execute a class member function, just use the `()` operator on the member function of the object instance.

```
$o:=cs.myClass.new()
$o.who:="World"
$message:=$o.myClass.hello()
//$message: "Hello World"
```

Optionally, use the `Class constructor` keyword to declare properties of the object.

```
//in the Rectangle.4dm file
Class constructor ($width : Integer; $height : Integer)
    This.height:=$height
    This.width:=$width
    This.name:="Rectangle"
```

A class can extend another class by using `Class extends <ClassName>`. Superclasses can be called using the `Super` command. For example:

```
//in the Square.4dm file
Class extends rectangle

Class constructor ($length : Integer)
    // It calls the parent class's constructor with lengths
    // provided for the Rectangle's width and height
    Super($length;$length)

This.name:="Square"
```

## Operators

When you use the language, it is rare that you will simply want a piece of data. It is more likely that you will want to do something to or with that data. You perform such calculations with operators. Operators, in general, take two pieces of data and perform an operation on them that results in a new piece of data. You are already familiar with many operators. For example,  $1 + 2$  uses the addition (or plus sign) operator to add two numbers together, and the result is 3. This table shows some familiar numeric operators:

| <b>Operator</b> | <b>Operation</b> | <b>Example</b>        |
|-----------------|------------------|-----------------------|
| +               | Addition         | 1 + 2 results in<br>3 |
| -               | Subtraction      | 3 - 2 results in 1    |
| *               | Multiplication   | 2 * 3 results in 6    |
| /               | Division         | 6 / 2 results in 3    |

Numeric operators are just one type of operator available to you. 4D supports many different types of data, such as numbers, text, dates, and pictures, so there are operators that perform operations on these different data types.

The same symbols are often used for different operations, depending on the data type. For example, the plus sign (+) performs different operations with different data:

| <b>Data Type</b> | <b>Operation</b> | <b>Example</b>   |
|------------------|------------------|--|
| Number           | Addition         | 1 + 2 adds the numbers and results in 3  |
| String           | Concatenation    | "Hello " + "there" concatenates (joins together) the strings and results in "Hello there"            |
| Date and Number  | Date addition    | !1989-01-01! + 20 adds 20 days to the date January 1, 1989, and results in the date January 21, 1989 |

## Expressions

Simply put, expressions return a value. In fact, when using the 4D language, you use expressions all the time and tend to think of them only in terms of the value they represent. Expressions are also sometimes referred to as formulas.

Expressions are made up of almost all the other parts of the language: commands, operators, variables, fields, object properties, and collection elements. You use expressions to build statements (lines of code), which in turn are used to build methods. The language uses expressions wherever it needs a piece of data.

Expressions rarely “stand alone.” There are several places in 4D where an expression can be used by itself. It includes:

- Formula editor (apply formula, query with formula, order by formula)
- The `EXECUTE FORMULA` command
- The Property list, where an expression can be used as a data source for most of widgets
- Debugger where the value of expressions can be checked
- Quick Report editor as a formula for a column

## Expression types

You refer to an expression by the data type it returns. There are several expression types. The following table gives examples of each type of expression.

| Expression                  | Type               | Description   |
|-----------------------------|--------------------|---|
| "Hello"                     | String             | The word Hello is a string constant, indicated by the double quotation marks.   |
| "Hello " + "there"          | String             | Two strings, "Hello " and "there", are added together (concatenated) with the string concatenation operator (+). The string "Hello there" is returned.                          |
| "Mr. " + [People]Name       | String             | Two strings are concatenated: the string "Mr. " and the current value of the Name field in the People table. If the field contains "Smith", the expression returns "Mr. Smith". |
| Uppercase("smith")          | String             | This expression uses <code>Uppercase</code> , a command from the language, to convert the string "smith" to uppercase. It returns "SMITH".                                      |
| 4                           | Number             | This is a number constant, 4.   |
| 4 * 2                       | Number             | Two numbers, 4 and 2, are multiplied using the multiplication operator (*). The result is the number 8.   |
| myButton                    | Number             | This is a variable associated to a button. It returns the current value of the button: 1 if it was clicked, 0 if not.   |
| !1997-01-25!                | Date               | This is a date constant for the date 1/25/97 (January 25, 1997).  |
| Current date + 30           | Date               | This is a date expression that uses the <code>Current date</code> command to get today's date. It adds 30 days to today's date and returns the new date.                        |
| ?8:05:30?                   | Time               | This is a time constant that represents 8 hours, 5 minutes, and 30 seconds.   |
| ?2:03:04? + ?<br>1:02:03?   | Time               | This expression adds two times together and returns the time 3:05:07.   |
| True                        | Boolean            | This command returns the Boolean value TRUE.  |
| 10 # 20                     | Boolean            | This is a logical comparison between two numbers. The number sign (#) means "is not equal to". Since 10 "is not equal to" 20, the expression returns TRUE.                      |
| "ABC" = "XYZ"               | Boolean            | This is a logical comparison between two strings. They are not equal, so the expression returns FALSE.  |
| My Picture + 50             | Picture            | This expression takes the picture in My Picture, moves it 50 pixels to the right, and returns the resulting picture.  |
| ->[People]Name              | Pointer            | This expression returns a pointer to the field called [People]Name.   |
| Table (1)                   | Pointer            | This is a command that returns a pointer to the first table.  |
| JSON Parse<br>(MyString)    | Object             | This is a command that returns MyString as an object (if proper format)   |
| JSON Parse<br>(MyJSONArray) | Collection         | This is a command that returns MyJSONArray as a collection (if proper format)   |
| Form.pageNumber             | Object property    | An object property is an expression that can be of any supported type   |
| Col[5]                      | Collection element | A collection element is an expression that can be of any supported type   |
| \$entitySel[0]              | Entity             | A element of an ORDA entity selection is an expression of the entity type. This kind of expression is <b>non-assignable</b>   |

## Assignable vs non-assignable expressions

An expression can simply be a literal constant, such as the number 4 or the string

"Hello", or a variable like `$myButton`. It can also use operators. For example, `4 + 2` is an expression that uses the addition operator to add two numbers together and return the result 6. In any cases, these expressions are **non-assignable**, which means that you cannot assign a value to them. In 4D, expressions can be **assignable**. An expression is assignable when it can be used on the left side of an assignation. For example:

```
//$myVar variable is assignable, you can write:  
$myVar:="Hello" //assign "Hello" to myVar  
//Form.pageNumber is assignable, you can write:  
Form.pageNumber:=10 //assign 10 to Form.pageNumber  
//Form.pageTotal-Form.pageNumber is not assignable:  
Form.pageTotal- Form.pageNumber:=10 //error, non-assignable
```

In general, expressions that use an operator are non-assignable. For example, `[Person]FirstName+" "+[Person]LastName` is not assignable.

## Pointers

The 4D language provides an advanced implementation of pointers, that allow writing powerful and modular code. You can use pointers to reference tables, fields, variables, arrays, and array elements.

A pointer to an element is created by adding a "`->`" symbol before the element name, and can be dereferenced by adding the "`->`" symbol after the pointer name.

```
MyVar:="Hello"  
MyPointer:=->MyVar  
ALERT (MyPointer->)
```

## Code on several lines

You can write a single statement on several lines by terminating each line of the statement with a trailing backslash `\` character. The 4D language will consider all the lines at once. For example, both the following statements are equivalent:

```
$str:=String("hello world!")  
$str:=String("hello"+\  
" world"+\  
+"!")
```

## Comments

Comments are inactive lines of code. These lines are not interpreted by the 4D language and are not executed when the code is called.

There are two ways to create comments:

- `//` for single line comments
- `/* ... */` for inline or multiline comments.

Both styles of comments can be used simultaneously.

### Single line comments (`//comment`)

Insert `//` at the beginning of a line or after a statement to add a single line comment. Example:

```
//This is a comment
For($vCounter;1;100) //Starting loop
    //comment
    //comment
    //comment
End for
```

## Inline or multiline comments (`/*comment*/`)

Surround contents with `/* ... */` characters to create inline comments or multiline comment blocks. Both inline and multiline comment blocks begin with `/*` and end with `*/`.

- **Inline comments** can be inserted anywhere in the code. Example:

```
For /* inline comment */ ($vCounter;1;100)
    ...
End for
```

- **Multiline comment blocks** allows commenting an unlimited number of lines. Comment blocks can be nested (useful since the 4D code editor supports block collapsing). Example:

```
For ($vCounter;1;100)
/*
comments
/*
    other comments
*/
*/
...
End for
```

## Escape sequences

The 4D language allows you to use escape sequences (also called escape characters). An escape sequence is a sequence of characters that can be used to replace a "special" character.

The sequence consists of a backslash `\`, followed by a character. For instance, `\t` is an escape sequence for the **Tab** character. Escape sequences facilitate the entry of special characters: the previous example (`\t`) replaces the entry "Character(Tab)".

In 4D, the following escape sequences can be used:

### Escape sequence Character replaced

|                  |                            |
|------------------|----------------------------|
| <code>\n</code>  | LF (Line feed)             |
| <code>\t</code>  | HT (Tab)                   |
| <code>\r</code>  | CR (Carriage return)       |
| <code>\\"</code> | <code>\</code> (Backslash) |
| <code>\"</code>  | " (Quotation marks)        |

It is possible to use either upper or lower case in escape sequences.

In the following example, the **Carriage return** character (escape sequence `\r`) is inserted in a statement in order to obtain a dialog box:

```
ALERT("The operation has been completed successfully.\rYou may now
disconnect.")
```



# Operators

An operator is a symbol or a group of symbols that you use to check, modify, or combine values. You are already familiar with many operators. For example, `1 + 2` uses the addition (or plus sign) operator to add two numbers together, and the result is 3. Comparison operators, like `=` or `>`, let you compare two or more values.

The 4D language supports the operators you may already know from other languages like C or JavaScript. However, the assignment operator is `:=` to prevent it from being mistakenly used when the equal to operator (`=`) is intended. [Basic operators](#) such as arithmetic operators (`+`, `-`, `*`, `/`, `%`...) and comparison operators (`=`, `>`, `>=`...) can be used with numbers, but also with boolean, text, date, time, pointer, or picture data types. Like JavaScript, the 4D language supports the concept of [truthy and falsy values](#), which is used in [short-circuit operators](#).

## Terminology

The 4D language supports **binary** and **ternary** operators:

- binary operators operate on two targets (such as `2 + 3`) and appear in between their two targets.
- ternary operators operate on three targets. Like C, 4D has only one ternary operator, the [ternary conditional operator](#) (`a ? b : c`).

The values that operators affect are operands. In the expression `1 + 2`, the `+` symbol is a binary operator and its two operands are the values 1 and 2.

## Assignment operator

The **assignment operator** (`a := b`) initializes or updates the value of `a` with the value of `b`:

```
$myNumber:=3 //assigns 3 to MyNumber variable  
$myDate:=!2018/01/21! //assigns a date literal  
$myLength:=Length("Acme") //assigns the result of the command (4) to  
$myLength  
$col:=New collection // $col is initialized with an empty collection
```

Do NOT confuse the assignment operator `:=` with the equality comparison operator `=`. A different assignment operator (and not `=`) was deliberately chosen to avoid issues and confusion which often occur with `==` or `====` in other programming languages. Such errors are often difficult to recognize by the compiler and lead to time-consuming troubleshooting.

## Basic operators

Operator results depend on the **data types** they are applied to. 4D supports different operators on scalar data types. They are described with the data types, in the following sections:

- [Logical operators](#) (on **boolean** expressions)
- [Date operators](#)
- [Time operators](#)
- [Number operators](#)
- [Bitwise operators](#) (on **long integer** expressions)

- [Picture operators](#)
- [Pointer operators](#)
- [String operators](#)
- [Null operators](#)
- [Undefined operators](#)

## Compound assignment operators

- History

4D provides **compound assignment operators** that combine assignment with another operation. One example is the addition assignment operator (`+=`):

```
$a:=1  
$a+=2 // $a=3
```

The following compound assignment operators are supported:

| Operator       | Syntax             | Assigns | Example   |
|----------------|--------------------|---------|---|
| Addition       | Text += Text       | Text    | <code>\$t+=" World" // \$t:=\$t+" World"</code>                                     |
|                | Number += Number   | Number  | <code>\$n+=5 // \$n:=\$n+5</code>   |
|                | Date += Number     | Date    | <code>\$d+=5 // \$d:=\$d+5</code>   |
|                | Time += Time       | Time    | <code>\$t1+=\$t2 // \$t1:=\$t1+\$t2</code>  |
|                | Time += Number     | Number  | <code>\$t1+=5 // \$t1:=\$t1+5</code>  |
|                | Picture += Picture | Picture | <code>\$p1+=\$p2 // \$p1:=\$p1+\$p2 (add \$p2 to the right of \$p1)</code>          |
|                | Picture += Number  | Picture | <code>\$p1+=5 // \$p1:=\$p1+5 (move \$p1 horizontally 5 pixels to the right)</code> |
|                | Number -= Number   | Number  | <code>\$n-=5 // \$n:=\$n-5</code>   |
|                | Date -= Number     | Date    | <code>\$d-=5 // \$d:=\$d-5</code>   |
|                | Time -= Time       | Time    | <code>\$t1-=\$t2 // \$t1:=\$t1-\$t2</code>  |
| Subtraction    | Time -= Number     | Number  | <code>\$t1-=5 // \$t1:=\$t1-5</code>  |
|                | Picture -= Number  | Picture | <code>\$p1-=5 // \$p1:=\$p1-5 (move \$p1 horizontally 5 pixels to the left)</code>  |
|                | Number /= Number   | Number  | <code>\$n/=5 // \$n:=\$n/5</code>   |
|                | Time /= Time       | Time    | <code>\$t1/=5 // \$t1:=\$t1/\$t2</code>   |
|                | Time /= Number     | Number  | <code>\$t1/=5 // \$t1:=\$t1/5</code>  |
|                | Picture /= Picture | Picture | <code>\$p1+=\$p2 // \$p1:=\$p1+\$p2 (add \$p2 to the bottom of \$p1)</code>         |
|                | Picture /= Number  | Picture | <code>\$p1/=5 // \$p1:=\$p1/5 (move \$p1 vertically 5 pixels)</code>                |
|                | Text *= Number     | Text    | <code>\$t*="abc" // \$t:=\$t*"abc"</code>   |
|                | Number *= Number   | Number  | <code>\$n*=5 // \$n:=\$n*5</code>   |
|                | Time *= Time       | Time    | <code>\$t1*=\$t2 // \$t1:=\$t1*\$t2</code>  |
| Multiplication | Time *= Number     | Number  | <code>\$t1*=5 // \$t1:=\$t1*5</code>  |
|                | Picture *= Number  | Picture | <code>\$p1*=5 // \$p1:=\$p1*5 (resize \$p1 by 5)</code>                             |
|                |                    |         |   |

These operators apply on any [assignable expressions](#) (except pictures as object properties or collection elements).

The operation "source `operator` value" is not strictly equivalent to "source `:=` source `operator` value" because the expression designating the source (variable, field, object property, collection element) is only evaluated once. For example, in such expression as `getPointer() ->+=1` the `getPointer` method is called only once.

[Character indexing in text](#) and [byte indexing in blob](#) do not support these

operators.

## Examples

```
// Addition
$x:=2
$x+=5 // $x=7

$t:="Hello"
$t+=" World" // $t="Hello World"

$d:=!2000-11-10!
$d+=10 // $d=!2000-11-20!

// Subtraction
$x1:=10
$x1-=5 // $x1=5

$d1:=!2000-11-10!
$d1-=10 // $d1=!2000-10-31!

// Division
$x3:=10
$x3/=2 // $x3=5

// Multiplication
$x2:=10
$x2*=5 // $x2=10

$t2:="Hello"
$t2*=2 // $t2="HelloHello"
```

## Short-circuit operators

The **&&** and **||** operators are **short circuit operators**. A short circuit operator is one that doesn't necessarily evaluate all of its operands.

The difference with the single [& and | boolean operators](#) is that the short-circuit operators **&&** and **||** don't return a boolean value. They evaluate expressions as [truthy or falsy](#), then return one of the expressions.

### Short-circuit AND operator (**&&**)

- History

The rule is as follows:

Given `Expr1 && Expr2`:

The short-circuit AND operator evaluates operands from left to right, returning immediately with the value of the first falsy operand it encounters; if all values are [truthy](#), the value of the last operand is returned.

The following table summarizes the different cases for the **&&** operator:

#### **Expr1** **Expr2** **Value returned**

|        |        |       |
|--------|--------|-------|
| truthy | truthy | Expr2 |
| truthy | falsy  | Expr2 |
| falsy  | truthy | Expr1 |
| falsy  | falsy  | Expr1 |

## Example 1

```
var $v : Variant  
  
$v:= "Hello" && "World" // "World"  
$v:=False && 0 // False  
$v:=0 && False // False  
$v:=5 && !00-00-00! // 00/00/00  
$v := 5 && 10 && "hello" // "hello"
```

## Example 2

Say you have an online store, and some products have a tax rate applied, and others don't.

To calculate the tax, you multiply the price by the tax rate, which may not have been specified.

So you can write this:

```
var $tax : Variant  
  
$tax:=$item.taxRate && ($item.price*$item.taxRate)
```

`$tax` will be NULL if `taxRate` is NULL (or undefined), otherwise it will store the result of the calculation.

## Example 3

Short-circuit operators are useful in tests such as:

```
If (($myObject#Null) && ($myObject.value>10))  
    //code  
End if
```

If `$myObject` is Null, the second argument is not executed, thus no error is thrown.

## Short-circuit OR operator (||)

- History

The `||` operator returns the value of one of the specified operands. The expression is evaluated left to right and tested for possible "short-circuit" evaluation using the following rule:

Given `Expr1 || Expr2`:

If `Expr1` is [truthy](#), `Expr2` is not evaluated and the calculation returns `Expr1`.

If `Expr1` is [falsy](#), the calculation returns `Expr2`.

The following table summarizes the different cases and the value returned for the `||` operator:

### Expr1 Expr2 Value returned

|        |        |       |
|--------|--------|-------|
| truthy | truthy | Expr1 |
| truthy | falsy  | Expr1 |
| falsy  | truthy | Expr2 |
| falsy  | falsy  | Expr2 |

## Example 1

Say you have a table called Employee. Some employees have entered a phone number, and others haven't. This means that `$emp.phone` could be NULL, and you cannot assign NULL to a Text variable. But you can write the following:

```
var $phone : Text  
$phone:=$emp.phone || "n/a"
```

In which case `$phone` will store either a phone number or the "n/a" string.

## Example 2

Given a table called Person with a *name* field, as well as a *maiden name* field for married women.

The following example checks if there is a maiden name and stores it in a variable, otherwise it simply stores the person's name:

```
var $name: Text  
$name:=$person.maidenName || $person.name
```

## Precedence

The `&&` and `||` operators have the same precedence as the logical operators `&` and `||`, and are evaluated left to right.

This means that `a || b && c` is evaluated as `(a || b) && c`.

## Ternary operator

- History

The ternary conditional operator allows you to write one-line conditional expressions. For example, it can replace a full sequence of [If...Else](#) statements.

It takes three operands in the following order:

- a condition followed by a question mark (?)
- an expression to execute if the condition is [truthy](#), followed by a colon (:)
- an expression to execute if the condition is [falsy](#)

## Syntax

The syntax is as follows:

```
condition ? exprIfTruthy : exprIfFalsy
```

Since the [token syntax](#) uses colons, we recommend inserting a space after the colon `:` or enclosing tokens using parentheses to avoid any conflicts.

## Examples

### A simple example

```

var $age : Integer
var $beverage : Text

$age:=26
$beverage:=($age>=21) ? "Beer" : "Juice"

ALERT($beverage) // "Beer"

```

## Handling data from a table

This example stores a person's full name in a variable, and handles the case when no first name or last name has been specified:

```

var $fullname : Text

// If one of the names is missing, store the one that exists, otherwise
store an empty string
$fullname:=($person.firstname && $person.lastname) ?
($person.firstname+" "+$person.lastname) : ($person.lastname || 
$person.firstname) || ""

```

## Truthy and falsy

- History

As well as a type, each value also has an inherent Boolean value, generally known as either **truthy** or **falsy**.

**truthy** and **falsy** values are only evaluated by [short-circuit](#) and [ternary](#) operators.

The following values are **falsy**:

- false
- Null
- undefined
- Null object
- Null collection
- Null pointer
- Null picture
- Null date !00-00-00!
- "" - Empty strings
- [] - Empty collections
- {} - Empty objects

All other values are considered **truthy**, including:

- 0 - numeric zero (Integer or otherwise)

In 4D, **truthy** and **falsy** evaluation reflects the **usability** of a value, which means that a truthy value exists and can be processed by the code without generating errors or unexpected results. The rationale behind this is to provide a convenient way to handle *undefined* and *null* values in objects and collections, so that a reduced number of [If...Else](#) statements are necessary to avoid runtime errors.

For example, when you use a [short-circuit OR operator](#):

```
$value:=$object.value || $defaultValue
```

... you get the default value whenever *\$object* does not contain the `value` property

OR when it is *null*. So this operator checks the existence or usability of the value instead of a specific value. Note that because the numerical value 0 exists and is usable, it is not treated specially, thus it is **truthy**.

Regarding values representing collections, objects, or strings, "empty" values are considered **falsy**. It is handy when you want to assign a default value whenever an empty one is encountered.

```
$phone := $emp.phone || "n/a"
```

## Data types overview

In 4D, data are handled according to their type in two places: database fields and the 4D language.

Although they are usually equivalent, some data types available at the database level are not directly available in the language and are automatically converted. Conversely, some data types can only be handled through the language. The following table lists all available data types and how they are supported/declared:

| <b>Data Types</b>               | <b>Database support(1)</b> | <b>Language support</b> | <b><u>var</u><br/>declaration</b> | <b><u>C</u> or <u>ARRAY</u><br/>declaration</b> |
|---------------------------------|----------------------------|-------------------------|-----------------------------------|---|
| <a href="#">Alphanumeric</a>    | Yes                        | Converted to text       | -                                 | -   |
| <a href="#">Text</a>            | Yes                        | Yes                     | Text                              | C_TEXT, ARRAY TEXT                              |
| <a href="#">Date</a>            | Yes                        | Yes                     | Date                              | C_DATE, ARRAY DATE                              |
| <a href="#">Time</a>            | Yes                        | Yes                     | Time                              | C_TIME, ARRAY TIME                              |
| <a href="#">Boolean</a>         | Yes                        | Yes                     | Boolean                           | C_BOOLEAN, ARRAY BOOLEAN                        |
| <a href="#">Integer</a>         | Yes                        | Converted to longint    | Integer                           | ARRAY INTEGER                                   |
| <a href="#">Longint</a>         | Yes                        | Yes                     | Integer                           | C_LONGINT, ARRAY LONGINT                        |
| <a href="#">Longint 64 bits</a> | Yes (SQL)                  | Converted to real       | -                                 | -   |
| <a href="#">Real</a>            | Yes                        | Yes                     | Real                              | C_REAL, ARRAY REAL                              |
| <a href="#">Undefined</a>       | -                          | Yes                     | -                                 | -   |
| <a href="#">Null</a>            | -                          | Yes                     | -                                 | -   |
| <a href="#">Pointer</a>         | -                          | Yes                     | Pointer                           | C_POINTER, ARRAY POINTER                        |
| <a href="#">Picture</a>         | Yes                        | Yes                     | Picture                           | C_PICTURE, ARRAY PICTURE                        |
| <a href="#">BLOB</a>            | Yes                        | Yes                     | Blob,<br>4D.Blob                  | C_BLOB, ARRAY BLOB                              |
| <a href="#">Object</a>          | Yes                        | Yes                     | Object                            | C_OBJECT, ARRAY OBJECT                          |
| <a href="#">Collection</a>      | -                          | Yes                     | Collection                        | C_COLLECTION                                    |
| <a href="#">Variant(2)</a>      | -                          | Yes                     | Variant                           | C_VARIANT                                       |

(1) Note that ORDA handles database fields through objects (entities) and thus, only supports data types available to these objects. For more information, see the [Object](#) data type description.

(2) Variant is actually not a *data* type but a *variable* type that can contain a value of any other data type.

## Default values

When [variables](#) or [parameters](#) are typed by means of an [explicit declaration](#), they receive a default value, which they will keep during the session as long as they have not been assigned.

The default value depends on the variable type:

| Type       | Default value  |
|------------|----------------|
| Booleen    | False          |
| Date       | 00-00-00       |
| Longint    | 0              |
| Time       | 00:00:00       |
| Picture    | picture size=0 |
| Real       | 0              |
| Pointer    | Nil=true       |
| Text       | ""             |
| Blob       | Blob size=0    |
| Object     | null           |
| Collection | null           |
| Variant    | undefined      |

## Converting data types

The 4D language contains operators and commands to convert between data types, where such conversions are meaningful. The 4D language enforces data type checking. For example, you cannot write: "abc"+0.5+!12/25/96!-?00:30:45?. This will generate syntax errors.

The following table lists the basic data types, the data types to which they can be converted, and the commands used to do so:

### Data Type to Convert to String to Number to Date to Time to Boolean

|            |        |      |      |      |
|------------|--------|------|------|------|
| String (1) | Num    | Date | Time | Bool |
| Number (2) | String |      |      | Bool |
| Date       | String |      |      | Bool |
| Time       | String |      |      | Bool |
| Boolean    | Num    |      |      |      |

(1) Strings formatted in JSON can be converted into scalar data, objects, or collections, using the `JSON Parse` command.

(2) Time values can be treated as numbers.

**Note:** In addition to the data conversions listed in this table, more sophisticated data conversions can be obtained by combining operators and other commands.

## BLOB

A BLOB (Binary Large OBject) field, variable or expression is a contiguous series of bytes that can be treated as one whole object, or whose bytes can be addressed individually.

A blob is loaded into memory in its entirety. A blob variable is held and exists in memory only. A blob field is loaded into memory from the disk, like the rest of the record to which it belongs.

Like other field types that can retain a large amount of data (such as the Picture field type), Blob fields are not duplicated in memory when you modify a record.

Consequently, the result returned by the `Old` and `Modified` commands is not significant when applied to a Blob field.

## Blob Types

Using the 4D language, there are two ways to handle a blob:

- **as a scalar value:** a blob can be stored in a Blob variable or field and altered.
- **as an object (`4D.Blob`):** a `4D.Blob` is a blob object. You can encapsulate a blob or part of it in a `4D.Blob` without altering the original blob. This method is called [boxing](#). For more info on how to instantiate a `4D.Blob`, see [Blob Class](#).

Each blob type has its advantages. Use the following table to determine which one suits your needs:

|                                      | Blob       |        |
|--------------------------------------|------------|--------|
|                                      | 4D.Blob    | Scalar |
| Alterable                            | Yes        | No     |
| Shareable in objects and collections | No         | Yes    |
| Passed by reference*                 | No         | Yes    |
| Performance when accessing bytes     | +          | -      |
| Maximum size                         | 2GB Memory |        |

\*Unlike the 4D commands designed to take a scalar blob as a parameter, passing a scalar blob to a method duplicates it in memory. When working with methods, using blob objects (`4D.Blob`) is more efficient, as they are passed by reference.

By default, 4D sets the maximum size of scalar blobs to 2GB, but this size limit may be lower depending on your OS and how much space is available.

You cannot use operators on blobs.

## Checking if a variable holds a scalar blob or a `4D.Blob`

Use the [Value type](#) command to determine if a value is of type Blob or Object. To check that an object is a blob object (`4D.Blob`), use [OB instance of](#):

```
var $myBlob: Blob
var $myBlobObject: 4D.Blob
$myBlobObject:=4D.Blob.new()

$type:= Value type($myblobObject) // 38 (object)
$is4DBlob:= OB Instance of($myblobObject; 4D.Blob) //True
```

## Passing blobs as parameters

Scalar blobs and blob objects can be passed as parameters to 4D commands or plug-in routines that expect blob parameters.

### Passing blobs and blob objects to 4D commands

You can pass a scalar blob or a `4D.Blob` to any 4D command that takes a blob as a parameter:

```
var $myBlob: 4D.Blob  
CONVERT FROM TEXT("Hello, World!"; "UTF-8"; $myBlob)  
$myText:= BLOB to text( $myBlob ; UTF8 text without length )
```

Some 4D commands alter the original blob, and thus do not support the `4D.Blob` type:

- [DELETE FROM BLOB](#)
- [INSERT IN BLOB](#)
- [INTEGER TO BLOB](#)
- [LONGINT TO BLOB](#)
- [REAL TO BLOB](#)
- [SET BLOB SIZE](#)
- [TEXT TO BLOB](#)
- [VARIABLE TO BLOB](#)
- [LIST TO BLOB](#)
- [SOAP DECLARATION](#)
- [WEB SERVICE SET PARAMETER](#)

### Passing blobs and blob objects to methods

You can pass blobs and blob objects (`4D.Blob`) to methods. Keep in mind that unlike blob objects, which are passed by reference, scalar blobs are duplicated in memory when passed to methods.

### Passing a scalar blob by reference using a pointer

To pass a scalar blob to your own methods without duplicating it in memory, define a pointer to the variable that stores it and pass the pointer as a parameter.

#### Examples:

```
// Declare a variable of type Blob  
var $myBlobVar: Blob  
// Pass the blob as parameter to a 4D command  
SET BLOB SIZE($myBlobVar;1024*1024)  
// Pass the blob as parameter to an external routine  
$errCode:=Do Something With This blob($myBlobVar)  
// Pass the blob as a parameter to a method that returns a blob  
var $retrieveBlob: Blob  
retrieveBlob:=Fill_Blob($myBlobVar)  
// Pass a pointer to the blob as a parameter to your own method,  
COMPUTE BLOB(->$myBlobVar)
```

**Note for Plug-in developers:** A BLOB parameter is declared as "&O" (the letter "O", not the digit "0").

### Assigning a blob variable to another

You can assign a Blob variable to another:

## **Example:**

```
// Declare two variables of type Blob
var $vBlobA; $vBlobB : Blob
// Set the size of the first blob to 10K
SET BLOB SIZE($vBlobA;10*1024)
// Assign the first blob to the second one
$vBlobB:=$vBlobA
```

## **Automatic conversion of blob type**

4D automatically converts scalar blobs to blob objects, and vice versa, when they're assigned to each other. For example:

```
// Create a variable of type Blob and an object variable
var $myBlob: Blob
var $myObject : Object

// Assign that blob to a property of $myObject named "blob"
$myObject:=New object("blob"; $myBlob)

// The blob stored in $myBlob is automatically converted to a 4D.Blob
$type:= OB Instance of($myObject.blob; 4D.Blob) //True

// Conversion from 4D.Blob to Blob
$myBlob:= $myObject.blob
$type:= Value type($myBlob) // Blob
```

When converting a `4D.Blob` to a scalar blob, if the size of the `4D.Blob` exceeds the maximum size for scalar blobs, the resulting scalar blob is empty. For example, when the maximum size for scalar blobs is 2GB, if you convert a `4D.Blob` of 2.5GB to a scalar blob, you obtain an empty blob.

## **Modifying a scalar blob**

Unlike blob objects, scalar blobs can be altered. For example:

```
var $myBlob : Blob
SET BLOB SIZE ($myBlob ; 16*1024)
```

## **Individually accessing bytes in a blob**

### **Accessing a scalar blob's bytes**

You can access individual bytes of a scalar blob using curly brackets `{ }`. Within a blob, bytes are numbered from 0 to N-1, where N is the size of the BLOB:

```
// Declare a variable of type Blob
var $vBlob : Blob
// Set the size of the blob to 256 bytes
SET BLOB SIZE($vBlob;256)
// The following code loops through the blob to set each byte to zero
For(vByte;0;BLOB size($vBlob)-1)
    $vBlob{vByte}:=0
End for
```

Since you can address all the bytes of a blob individually, you can store whatever you want in a Blob variable or field.

### **Accessing a `4D.Blob`'s bytes**

Use square brackets `[]` to directly access a specific byte in a `4D.Blob`

```
var $myBlob: 4D.Blob  
CONVERT FROM TEXT("Hello, World!"; "UTF-8"; $myBlob)  
$myText:= BLOB to text ( $myBlob ; UTF8 text without length )  
$byte:=$myBlob[5]
```

Since a `4D.Blob` cannot be altered, you can read the bytes of a `4D.Blob` using this syntax, but not modify them.

## Boolean

A boolean field, variable or expression can be either TRUE or FALSE.

## Boolean functions

4D provides the Boolean functions `True`, `False`, and `Not` in the dedicated **Boolean** theme. For more information, see the descriptions of these commands.

### Example

This example sets a Boolean variable based on the value of a button. It returns True in `myBoolean` if the `myButton` button was clicked and False if the button was not clicked. When a button is clicked, the button variable is set to 1.

```
If (myButton=1) //If the button was clicked  
    myBoolean:=True //myBoolean is set to True  
Else //If the button was not clicked,  
    myBoolean:=False //myBoolean is set to False  
End if
```

The previous example can be simplified into one line.

```
myBoolean:=(myButton=1)
```

## Logical operators

4D supports two logical operators that work on Boolean expressions: conjunction (AND) and inclusive disjunction (OR). A logical AND returns TRUE if both expressions are TRUE. A logical OR returns TRUE if at least one of the expressions is TRUE. The following table shows the logical operators:

| Operation | Syntax            | Returns | Expression             | Value |
|-----------|-------------------|---------|------------------------|-------|
| AND       | Boolean & Boolean | Boolean | ("A" = "A") & (15 # 3) | True  |
|           |                   |         | ("A" = "B") & (15 # 3) | False |
|           |                   |         | ("A" = "B") & (15 = 3) | False |
| OR        | Boolean   Boolean | Boolean | ("A" = "A")   (15 # 3) | True  |
|           |                   |         | ("A" = "B")   (15 # 3) | True  |
|           |                   |         | ("A" = "B")   (15 = 3) | False |

The following is the truth table for the AND logical operator:

### Expr1 Expr2 Expr1 & Expr2

|       |       |       |
|-------|-------|-------|
| True  | True  | True  |
| True  | False | False |
| False | True  | False |
| False | False | False |

The following is the truth table for the OR logical operator:

## **Expr1 Expr2 Expr1 | Expr2**

|       |       |       |
|-------|-------|-------|
| True  | True  | True  |
| True  | False | True  |
| False | True  | True  |
| False | False | False |

**Tip:** If you need to calculate the exclusive disjunction between Expr1 and Expr2, evaluate:

```
(Expr1|Expr2) & Not(Expr1 & Expr2)
```

In boolean contexts, the 4D language also supports [short-circuit operators](#) (`&&` and `||`) and the [truthy and falsy](#) concept.

## Collection

Collections are ordered lists of values of similar or mixed types (text, number, date, object, boolean, collection, or null).

Collection type variables are managed using object notation (see [Syntax basics](#)).

To access a collection element, you need to pass the element number inside square brackets:

```
collectionRef [expression]
```

You can pass any valid 4D expression which returns a positive integer in *expression*. Examples:

```
myCollection[5] //access to 6th element of the collection  
myCollection[$var]
```

**Warning:** Collection elements are numbered from 0.

You can assign a value to a collection element or get a collection element value:

```
myCol[10]:="My new element"  
$myVar:=myCol[0]
```

If you assign an element's index that surpasses the last existing element of the collection, the collection is automatically resized and all new intermediary elements are assigned a null value:

```
var myCol : Collection  
myCol:=New collection("A"; "B")  
myCol[5]:="Z"  
//myCol[2]=null  
//myCol[3]=null  
//myCol[4]=null
```

## Instantiation

Collections must have been instantiated, otherwise trying to read or modify their elements will generate a syntax error.

Collection instantiation can be done in one of the following ways:

- using the [New collection](#) command,
- using the `[]` operator.



Several 4D commands and functions return collections, for example [Get Monitored Activity](#) or [collection.copy](#). In this case, it is not necessary to instantiate explicitly the collection, the 4D language does it for you.

### New collection command

The [New collection](#) command creates a new empty or prefilled collection and returns its reference.

Examples:

```

var $colVar : Collection //declaration of a collection type 4D
variable
$colVar:=New collection //instantiation of the collection and
assignment to the 4D variable

var $colFilled : Collection
$colFilled:=New collection("a";"b";1;42;{}) //instantiation and
assignment of a prefilled collection

```

## [ ] operator

The [ ] operator allows you to create a **collection literal**. A collection literal is a list of zero or more expressions, each of which represents a collection element, enclosed in square brackets ([ ]). When you create a collection using a collection literal, it is instantiated with the specified values as its elements, and its length is set to the number of arguments specified.

Since any element is considered an expression, you can create sub-collections using [ ] in elements. You can also create and reference **object literals**.

If an element is undefined, it will appear as Null in the collection.

Examples:

```

var $coll; $col2; $users : Collection
$coll:=[] //empty collection
$col2:=[1;2;3;4;5;6] //collection of numbers
//collection of objects
$users:=[{name: "Alice"; \
height: 183; \
eyecolor: "hazel"; \
id: $col2[5]\
}; \
{name: "Bob"; \
height: 172; \
eyecolor: "blue"\
}]

```

### NOTE

If you create a collection literal containing a single element, make sure you do not use a name corresponding to an existing table name, otherwise the table syntax [tableName] will take priority.

## Regular or shared collection

You can create two types of collections:

- regular (non-shared) collections, using the [New collection](#) command or collection literal syntax ([ ]). These collections can be edited without any specific access control but cannot be shared between processes.
- shared collections, using the [New shared collection](#) command. These collections can be shared between processes, including preemptive threads. Access to these collections is controlled by [Use....End use](#) structures.

For more information, refer to the [Shared objects and collections](#) section.

## Collection functions

4D collection references benefit from special class functions (sometimes named *member functions*). Collection functions are listed in the [Class API Reference](#) section.

For example:

```
$newCol:=$col.copy() //deep copy of $col to $newCol  
$col.push(10;100) //add 10 and 100 to the collection
```

Some functions return the original collection after modification, so that you can run the calls in a sequence:

```
$col:=New collection(5;20)  
$col2:=$col.push(10;100).sort() // $col2=[5,10,20,100]
```

## propertyPath parameter

Several functions accept a *propertyPath* as parameter. This parameter stands for:

- either an object property name, for example "lastName"
- or an object property path, i.e. a hierarchical sequence of sub-properties linked with dot characters, for example "employee.children.firstName".

**Warning:** When using functions and *propertyPath* parameters, you cannot use ".", "[", or spaces in property names since it will prevent 4D from correctly parsing the path:

```
$vmin:=$col.min("My.special.property") //undefined  
$vmin:=$col.min(["My.special.property"]) //error
```

## Date

A Date field, variable or expression can be in the range of 1/1/100 to 12/31/32,767.

Although the representation mode for dates by can work with dates up to the year 32 767, certain operations passing through the system impose a lower limit.

**Note:** In the 4D Language Reference manual, Date parameters in command descriptions are denoted as Date, except when marked otherwise.

## Date literals

A date literal constant is enclosed by exclamation marks (!…!). A date must be structured using the ISO format (!YYYY-MM-DD!). Here are some examples of date constants:

```
!1976-01-01!  
!2004-09-29!  
!2015-12-31!
```

A null date is specified by *!00-00-00!*.

**Tip:** The Code Editor includes a shortcut for entering a null date. To type a null date, enter the exclamation (!) character and press Enter.

### Notes:

- For compatibility reasons, 4D accepts two-digit years to be entered. A two-digit year is assumed to be in the 20th or 21st century based on whether it is greater or less than 30, unless this default setting has been changed using the `SET DEFAULT CENTURY` command.
- If you have checked the "Use regional system settings" option (see Methods Page), you must use the date format defined in your system. Generally, in a US environment, dates are entered in the form month/day/year, with a slash "/" separating the values.

## Date operators

| <b>Operation</b>         | <b>Syntax</b> | <b>Returns</b> | <b>Expression</b>  | <b>Value</b>              |
|--------------------------|---------------|----------------|--|---------------------------|
| Date difference          | Date - Date   | Number         | <code>!2017-01-20! - !2017-01-01!</code>   | 19                        |
| Day addition             | Date + Number | Date           | <code>!2017-01-20! + 9</code>  | <code>!2017-01-29!</code> |
| Day subtraction          | Date - Number | Date           | <code>!2017-01-20! - 9</code>  | <code>!2017-01-11!</code> |
| Equality                 | Date = Date   | Boolean        | <code>!2017-01-01! == !2017-01-01!</code><br><code>!2017-01-20! == !2017-01-01!</code>         | True<br>False             |
| Inequality               | Date # Date   | Boolean        | <code>!2017-01-20! != !2017-01-01!</code><br><code>!2017-01-20! != !2017-01-20!</code>         | True<br>False             |
| Greater than             | Date > Date   | Boolean        | <code>!2017-01-20! &gt; !2017-01-01!</code><br><code>!2017-01-20! &gt; !2017-01-20!</code>     | True<br>False             |
| Less than                | Date < Date   | Boolean        | <code>!2017-01-01! &lt; !2017-01-20!</code><br><code>!2017-01-20! &lt; !2017-01-20!</code>     | True<br>False             |
| Greater than or equal to | Date >= Date  | Boolean        | <code>!2017-01-20! &gt;= !2017-01-01!</code><br><code>!2017-01-01! &gt;= !2017-01-20!</code>   | True<br>False             |
| Less than or equal to    | Date ¥<= Date | Boolean        | <code>!2017-01-01! ¥&lt;= !2017-01-20!</code><br><code>!2017-01-20! ¥&lt;= !2017-01-01!</code> | True<br>False             |

## Null and Undefined

Null and Undefined are data types that handle cases where the value of an expression is not known.

### Null

Null is a special data type with only one possible value: **null**. This value is returned by an expression that does not contain any value.

In the 4D language and for object field attributes, null values are managed through the `Null` function. This function can be used with the following expressions for setting or comparing the null value:

- object attributes
- collection elements
- variables of the object, collection, pointer, picture, or variant type.

### Undefined

Undefined is not actually a data type. It denotes a variable that has not yet been defined. A function (a project method that returns a result) can return an undefined value if, within the method, the function result (\$0) is assigned an undefined expression (an expression calculated with at least one undefined variable). A field cannot be undefined (the `Undefined` command always returns False for a field). A variant variable has **undefined** as default value.

## Null operators

| Operation  | Syntax              | Returns | Expression                   | Value |
|------------|---------------------|---------|------------------------------|-------|
| Equality   | Null = Null         | Boolean | a.nullProp = b.nullProp      | True  |
|            | Null = Undefined    | Boolean | a.nullProp = b.undefinedProp | True  |
|            | Null = scalar value | Boolean | a.nullProp = 42              | False |
| Inequality | Null # Null         | Boolean | a.nullProp # b.nullProp      | False |
|            | Null # Undefined    | Boolean | a.nullProp # b.undefinedProp | False |
|            | Null # scalar value | Boolean | a.nullProp # 42              | True  |

*scalar values* are values of type string, Date, Time, Boolean, number, or Blob. When declared, their [default value](#) is neither undefined nor null. Other types (Pointer, Picture, Object, Collection) have undefined or null default value. Ex:

```
var $object : Object  
var $text : Text  
  
//$object = null  
//$text = ""
```



Comparisons with Greater than (`>`), Less than (`<`), Greater than or equal to (`>=`), and Less than or equal to (`<=`) operators are not supported with Null values and return an error.

# Undefined operators

| Operation             | Syntax   | Returns | Expression                               | Value |
|-----------------------|--|---------|--|-------|
| Equality              | Undefined $\equiv$ Undefined                         | Boolean | a.undefinedProp $\equiv$ b.undefinedProp | True  |
|                       | Undefined $\equiv$ Null                              | Boolean | a.undefinedProp $\equiv$ c.nullProp      | True  |
|                       | Undefined $\equiv$ other values                      | Boolean | a.undefinedProp $\equiv$ 42              | False |
| Inequality            | Undefined $\neq$ Undefined                           | Boolean | a.undefinedProp $\neq$ b.undefinedProp   | False |
|                       | Undefined $\neq$ Null                                | Boolean | a.undefinedProp $\neq$ b.nullProp        | False |
|                       | Undefined $\neq$ other values                        | Boolean | a.undefinedProp $\neq$ 42                | True  |
| Greater than          | Undefined $>$ string, Date, Time, Boolean, number    | Boolean | a.undefinedProp $>$ "abc"                | False |
|                       | Undefined $<$ string, Date, Time, Boolean, number    | Boolean | a.undefinedProp $<$ "abc"                | False |
|                       | Undefined $\geq$ string, Date, Time, Boolean, number | Boolean | a.undefinedProp $\geq$ "abc"             | False |
| Less than or equal to | Undefined $\leq$ string, Date, Time, Boolean, number | Boolean | a.undefinedProp $\leq$ "abc"             | False |
|                       | Undefined $\neq$ string, Date, Time, Boolean, number | Boolean | a.undefinedProp $\neq$ "abc"             | False |

*other values* are expressions of any type with a value neither Undefined nor Null.



Comparisons of Undefined values with Pointer, Picture, Blob, Object, Collection, Undefined or Null values using Greater than ( $>$ ), Less than ( $<$ ), Greater than or equal to ( $\geq$ ), and Less than or equal to ( $\leq$ ) operators are not supported and return an error.

## Examples

Here are the different results of the `Undefined` command as well as the `Null` command with object properties, depending on the context:

```
var $vEmp : Object
var $result : Boolean
$vEmp:=New object
$vEmp.name:="Smith"

$vEmp.children:=Null

$result:=Undefined($vEmp.name) //False
$result:=($vEmp.name=Null) //False

$result:=Undefined($vEmp.children) //False
$result:=($vEmp.children=NULL) //True

$result:=Undefined($vEmp.parent) //True
$result:=($vEmp.parent=NULL) //True
```

Examples of comparison results with undefined and null values:

```
var $result : Boolean
var $vObj : Object
var $vVar : Variant

$vObj:=New object
$vObj=null
//note that $vVar is not assigned

$result:=($vObj.undefined=42) //False
$result:=($vObj.undefined==$vObj.null) //True
$result:=($vObj.undefined==$vVar) //True

$result:=($vObj.undefined#$vObj.null) //False
$result:=($vObj.undefined#42) //True
$result:=($vObj.undefined#$vVar) //False

$result:=($vObj.undefined>"hello") //False
$result:=($vObj.undefined>$vVar) //Error
$result:=($vObj.undefined>$vObj.null) //Error
$result:=($vVar < 42) //False
```

## Number (Real, Longint, Integer)

Number is a generic term that stands for:

- Real field, variable or expression. The range for the Real data type is  $\pm 1.7e\pm 308$  (13 significant digits).
- Long Integer field, variable or expression. The range for the Long Integer data type (4-byte Integer) is  $-2^{31}..(2^{31}-1)$ .
- Integer field, array or expression. The range for the Integer data type (2-byte Integer) is  $-32,768..32,767$  ( $2^{15}..(2^{15}-1)$ ).

**Note:** Integer field values are automatically converted in Long integers when used in the 4D Language.

You can assign any Number data type to another; 4D does the conversion, truncating or rounding if necessary. However, when values are out of range, the conversion will not return a valid value. You can mix Number data types in expressions.

**Note:** In the 4D Language Reference manual, no matter the actual data type, the Real, Integer, and Long Integer parameters in command descriptions are denoted as number, except when marked otherwise.

## Number literals

A numeric literal constant is written as a real number. Here are some examples of numeric constants:

27  
123.76  
0.0076

The default decimal separator is a period (.), regardless of the system language. If you have checked the "Use regional system settings" option in the Methods Page of the Preferences, you must use the separator defined in your system.

Negative numbers are specified with the minus sign (-). For example:

-27  
-123.76  
-0.0076

## Number operators

| Operation                | Syntax           | Returns Expression Value                |
|--------------------------|------------------|---|
| Addition                 | Number + Number  | Number 2 + 3 5                          |
| Subtraction              | Number - Number  | Number 3 - 2 1                          |
| Multiplication           | Number * Number  | Number 5 * 2 10                         |
| Division                 | Number / Number  | Number 5 / 2 2.5                        |
| Longint division         | Number ¥ Number  | Number 5 ¥ 2 2                          |
| Modulo                   | Number % Number  | Number 5 % 2 1                          |
| Exponentiation           | Number ^ Number  | Number 2 ^ 3 8                          |
| Equality                 | Number = Number  | Boolean 10 = 10 True<br>10 = 11 False   |
| Inequality               | Number # Number  | Boolean 10 # 11 True<br>10 # 10 False   |
| Greater than             | Number > Number  | Boolean 11 > 10 True<br>10 > 11 False   |
| Less than                | Number < Number  | Boolean 10 < 11 True<br>11 < 10 False   |
| Greater than or equal to | Number >= Number | Boolean 11 >= 10 True<br>10 >= 11 False |
| Less than or equal to    | Number <= Number | Boolean 10 <= 11 True<br>11 <= 10 False |

The modulo operator % divides the first number by the second number and returns a whole number remainder. Here are some examples:

- 10 % 2 returns 0 because 10 is evenly divided by 2.
- 10 % 3 returns 1 because the remainder is 1.
- 10.5 % 2 returns 0 because the remainder is not a whole number.

### **WARNING:**

- The modulo operator % returns significant values with numbers that are in the Long Integer range (from minus  $2^{31}$  to  $2^{31}$  minus one). To calculate the modulo with numbers outside of this range, use the Mod command.
- The longint division operator ¥ returns significant values with integer numbers only.

### **Precedence**

The order in which an expression is evaluated is called precedence. 4D has a strict left-to-right precedence, in which algebraic order is not observed. For example:

`3+4*5`

returns 35, because the expression is evaluated as  $3 + 4$ , yielding 7, which is then multiplied by 5, with the final result of 35.

To override the left-to-right precedence, you MUST use parentheses. For example:

`3+(4*5)`

returns 23 because the expression  $(4 * 5)$  is evaluated first, because of the parentheses. The result is 20, which is then added to 3 for the final result of 23.

Parentheses can be nested inside other sets of parentheses. Be sure that each left parenthesis has a matching right parenthesis to ensure proper evaluation of

expressions. Lack of, or incorrect use of parentheses can cause unexpected results or invalid expressions. Furthermore, if you intend to compile your applications, you must have matching parentheses—the compiler detects a missing parenthesis as a syntax error.

## Bitwise operators

The bitwise operators operates on **Long Integer** expressions or values.

If you pass an Integer or a Real value to a bitwise operator, 4D evaluates the value as a Long Integer value before calculating the expression that uses the bitwise operator.

While using the bitwise operators, you must think about a Long Integer value as an array of 32 bits. The bits are numbered from 0 to 31, from right to left.

Because each bit can equal 0 or 1, you can also think about a Long Integer value as a value where you can store 32 Boolean values. A bit equal to 1 means **True** and a bit equal to 0 means **False**.

An expression that uses a bitwise operator returns a Long Integer value, except for the Bit Test operator, where the expression returns a Boolean value. The following table lists the bitwise operators and their syntax:

| Operation              | Operator | Syntax        | Returns              |
|------------------------|----------|---------------|----------------------|
| Bitwise AND            | &        | Long & Long   | Long                 |
| Bitwise OR (inclusive) |          | Long   Long   | Long                 |
| Bitwise OR (exclusive) | ¥^       | Long ¥^  Long | Long                 |
| Left Bit Shift         | <<       | Long << Long  | Long (see note 1)    |
| Right Bit Shift        | >>       | Long >> Long  | Long (see note 1)    |
| Bit Set                | ?+       | Long ?+ Long  | Long (see note 2)    |
| Bit Clear              | ?-       | Long ?- Long  | Long (see note 2)    |
| Bit Test               | ??       | Long ?? Long  | Boolean (see note 2) |

### Notes

1. For the `Left Bit Shift` and `Right Bit Shift` operations, the second operand indicates the number of positions by which the bits of the first operand will be shifted in the resulting value. Therefore, this second operand should be between 0 and 31. Note however, that shifting by 0 returns an unchanged value and shifting by more than 31 bits returns 0x00000000 because all the bits are lost. If you pass another value as second operand, the result is non-significant.
2. For the `Bit Set`, `Bit Clear` and `Bit Test` operations , the second operand indicates the number of the bit on which to act. Therefore, this second operand must be between 0 and 31; otherwise, the result of the expression is non-significant.

The following table lists the bitwise operators and their effects:

| Operation              | Description  |
|------------------------|--|
| Bitwise AND            | Each resulting bit is the logical AND of the bits in the two operands. Here is the logical AND table: <ul style="list-style-type: none"> <li>• 1 &amp; 1 --&gt; 1</li> <li>• 0 &amp; 1 --&gt; 0</li> <li>• 1 &amp; 0 --&gt; 0</li> <li>• 0 &amp; 0 --&gt; 0</li> </ul> In other words, the resulting bit is 1 if the two operand bits are 1; otherwise the resulting bit is 0. |
| Bitwise OR (inclusive) | Each resulting bit is the logical OR of the bits in the two operands. Here is the logical OR table: <ul style="list-style-type: none"> <li>• 1   1 --&gt; 1</li> <li>• 0   1 --&gt; 1</li> <li>• 1   0 --&gt; 1</li> <li>• 0   0 --&gt; 0</li> </ul> In other words, the resulting bit is 1 if at least one of the two operand bits is 1; otherwise the resulting bit is 0.    |
| Bitwise OR (exclusive) | Each resulting bit is the logical XOR of the bits in the two operands. Here is the logical XOR table: <ul style="list-style-type: none"> <li>• 1 ^  1 --&gt; 0</li> <li>• 0 ^  1 --&gt; 1</li> <li>• 1 ^  0 --&gt; 1</li> <li>• 0 ^  0 --&gt; 0</li> </ul> In other words, the resulting bit is 1 if only one of the two operand bits is 1; otherwise the resulting bit is 0.  |
| Left Bit Shift         | The resulting value is set to the first operand value, then the resulting bits are shifted to the left by the number of positions indicated by the second operand. The bits on the left are lost and the new bits on the right are set to 0. <b>Note:</b> Taking into account only positive values, shifting to the left by N bits is the same as multiplying by $2^N$ .       |
| Right Bit Shift        | The resulting value is set to the first operand value, then the resulting bits are shifted to the right by the number of position indicated by the second operand. The bits on the right are lost and the new bits on the left are set to 0. <b>Note:</b> Taking into account only positive values, shifting to the right by N bits is the same as dividing by $2^N$ .         |
| Bit Set                | The resulting value is set to the first operand value, then the resulting bit, whose number is indicated by the second operand, is set to 1. The other bits are left unchanged.  |
| Bit Clear              | The resulting value is set to the first operand value, then the resulting bit, whose number is indicated by the second operand, is set to 0. The other bits are left unchanged.  |
| Bit Test               | Returns True if, in the first operand, the bit whose number is indicated by the second operand is equal to 1. Returns False if, in the first operand, the bit whose number is indicated by the second operand is equal to 0.   |

## Examples

| <b>Operation</b>       | <b>Example</b>           | <b>Result</b> |
|------------------------|--------------------------|---------------|
| Bitwise AND            | 0x0000FFFF & 0xFF00FF00  | 0x0000FF00    |
| Bitwise OR (inclusive) | 0x0000FFFF   0xFF00FF00  | 0xFF00FFFF    |
| Bitwise OR (exclusive) | 0x0000FFFF ^  0xFF00FF00 | 0xFF0000FF    |
| Left Bit Shift         | 0x0000FFFF << 8          | 0x00FFFF00    |
| Right Bit Shift        | 0x0000FFFF >> 8          | 0x000000FF    |
| Bit Set                | 0x00000000 ?+ 16         | 0x00010000    |
| Bit Clear              | 0x00010000 ?- 16         | 0x00000000    |
| Bit Test               | 0x00010000 ?? 16         | True          |

# Object

Variables, fields or expressions of the Object type can contain various types of data. The structure of native 4D objects is based on the classic principle of "property/value" pairs. The syntax of these objects is based on JSON notation:

- A property name is always a text, for example "Name". It must follow [specific rules](#).
- A property value can be of the following type:
  - number (Real, Integer, etc.)
  - text
  - null
  - boolean
  - pointer (stored as such, evaluated using the `JSON Stringify` command or when copying),
  - date (date type or ISO date format string)
  - object(1) (objects can be nested on several levels)
  - picture(2)
  - collection

(1) ORDA objects such as [entities](#) or [entity selections](#) cannot be stored in **object fields**; however, they are fully supported in **object variables** in memory.

(2) When exposed as text in the debugger or exported to JSON, picture object properties print "[object Picture]".

## CAUTION

Keep in mind that property names differentiate between upper and lower case.

You manage Object type variables, fields or expressions using the [object notation](#) or the commands available in the **Objects (Language)** theme. Note that specific commands of the **Queries** theme such as `QUERY BY ATTRIBUTE`, `QUERY SELECTION BY ATTRIBUTE`, or `ORDER BY ATTRIBUTE` can be used to carry out processing on object fields.

Each property value accessed through the object notation is considered an expression. You can use such values wherever 4D expressions are expected:

- in 4D code, either written in the methods (Code Editor) or externalized (formulas, 4D tags files processed by `PROCESS 4D TAGS` or the Web Server, export files, 4D Write Pro documents...),
- in the Expression areas of the Debugger and the Runtime explorer,
- in the Property list of the Form editor for form objects: Variable or Expression field as well as various selection list box and columns expressions (Data Source, background color, style, or font color).

# Instantiation

Objects must have been instantiated, otherwise trying to read or modify their properties will generate a syntax error.

Object instantiation can be done in one of the following ways:

- using the [New object](#) command,
- using the `{ }` operator.

## ⓘ INFO

Several 4D commands and functions return objects, for example [Get\\_database](#), [measures](#) or [File](#). In this case, it is not necessary to instantiate explicitly the object, the 4D language does it for you.

### **New object command**

The [New object](#) command creates a new empty or prefilled object and returns its reference.

Examples:

```
var $obVar : Object //declaration of an object type 4D variable
$obVar:=New object //instantiation of an object and assignment to the
4D variable
```

```
var $obFilled : Object
$obFilled:=New object("name";"Smith";"age";42) //instantiation and
assignment of a prefilled object
```

### **{ } operator**

The `{ }` operator allows you to create an **object literal**. An object literal is a semi-column separated list of zero or more pairs of property names and associated values of an object, enclosed in curly braces (`{ }`). The object literal syntax creates empty or filled objects.

Since any property value is considered an expression, you can create sub-objects using `{ }` in property values. You can also create and reference **collection literals**.

Examples:

```
var $o ; $o2 ; $o3 : Object //declaration of object variables
$o := {} // instantiation of an empty object
$o2 := {a: "foo"; b: 42; c: {}; d: ($toto) ? true : false} //
instantiation of an object
// with properties {"a":"foo","b":42,"c":{}, "d":false})

// same properties using variables
var $a : Text
var $b : Number
var $c : Object
$a:="foo"
$b:=42
$c:={}
$o3:={ a: $a; b: $b; c: $c } // {"a":"foo";b:42;c:{}}
```

You can mix the [New object](#) and literal syntaxes:

```

$o:={ \
    ob1: {age: 42}; \
    ob2: New object("message"; "Hello"); \
    form1: Formula(return This.ob1.age+10); \
    form2 : Formula(ALERT($1)); \
    col: [1; 2; 3; 4; 5; 6]\
}

$o.form1() //52
$o.form2($o.ob2.message) // displays Hello
$col:=$o.col[5] //6

```

## Regular or shared object

You can create two types of objects:

- regular (non-shared) objects, using the [New object](#) command or object literal syntax (`{ }`). These objects can be edited without any specific access control but cannot be shared between processes.
- shared objects, using the [New shared object](#) command. These objects can be shared between processes, including preemptive threads. Access to these objects is controlled by `Use...End use` structures. For more information, refer to the [Shared objects and collections](#) section.

## Syntax basics

Object notation can be used to access object property values through a chain of tokens.

### Object properties

With object notation, object properties can be accessed in two ways:

- using a "dot" symbol:

```
object.propertyName
```

Example:

```
employee.name:="Smith"
```

- using a string within square brackets:

```
object["propertyName"]
```

Examples:

```

$vName:=employee["name"]
//or also:
$property:="name"
$vName:=employee[$property]

```

Since an object property value can be an object or a collection, object notation accepts a sequence of symbols to access sub-properties, for example:

```
$vAge:=employee.children[2].age
```

Object notation is available on any language element that can contains or returns an object, i.e:

- **Objects** themselves (stored in variables, fields, object properties, object arrays,

or collection elements). Examples:

```
$age:=$myObjVar.employee.age //variable  
$addr:=[Emp]data_obj.address //field  
$city:=$addr.city //property of an object  
$pop:=$aObjCountries{2}.population //object array  
$val:=$myCollection[3].subvalue //collection element
```

- **4D commands** that return objects. Example:

```
$measures:=Get database measures.DB.tables
```

- **Project methods** that return objects. Example:

```
// MyMethod1  
#DECLARE -> $o : Object  
$o:=New object("a";10;"b";20)  
  
//myMethod2  
$result:=MyMethod1.a //10
```

- **Collections** Example:

```
myColl.length //size of the collection
```

## Pointers

**Preliminary Note:** Since objects are always passed by reference, there is usually no need to use pointers. While just passing the object, internally 4D automatically uses a mechanism similar to a pointer, minimizing memory need and allowing you to modify the parameter and to return modifications. As a result, you should not need to use pointers. However, in case you want to use pointers, property values can be accessed through pointers.

Using object notation with pointers is very similar to using object notation directly with objects, except that the "dot" symbol must be omitted.

- Direct access:

```
pointerOnObject->propertyName
```

- Access by name:

```
pointerOnObject->["propertyName"]
```

Example:

```
var vObj : Object  
var vPtr : Pointer  
vObj:=New object  
vObj.a:=10  
vPtr:=->vObj  
x:=vPtr->a //x=10
```

## Null value

When using the object notation, the **null** value is supported though the **Null** command. This command can be used to assign or compare the null value to object properties or collection elements, for example:

```
myObject.address.zip:=Null  
If (myColl[2]=Null)
```

For more information, please refer to [Null and Undefined](#).

## Undefined value

Evaluating an object property can sometimes produce an undefined value. Typically when trying to read or assign undefined expressions, 4D will generate errors. This does not happen in the following cases:

- Reading a property of an undefined object or value returns undefined; assigning an undefined value to variables (except arrays) has the same effect as calling [CLEAR VARIABLE](#) with them:

```
var $o : Object
var $val : Integer
$val:=10 // $val=10
$val:=$o.a // $o.a is undefined (no error), and assigning this
value clears the variable
// $val=0
```

- Reading the **length** property of an undefined collection produces 0:

```
var $c : Collection // variable created but no collection is
defined
$c.size // $size = 0
```

- An undefined value passed as parameter to a project method is automatically converted to 0 or "" according to the declared parameter type.

```
var $o : Object
mymethod($o.a) // pass an undefined parameter

// In mymethod method
#Declare ($myText : Text) // parameter type is text
// $myText contains ""
```

- A condition expression is automatically converted to false when evaluating to undefined with the If and Case of keywords:

```
var $o : Object
If($o.a) // false
End if
Case of
    :($o.a) // false
End case
```

- Assigning an undefined value to an existing object property reinitializes or clears its value, depending on its type:
- Object, collection, pointer: Null
- Picture: Empty picture
- Boolean: False
- String: ""
- Number: 0
- Date: !00-00-00! if "Use date type instead of ISO date format in objects" setting is enabled, otherwise ""
- Time: 0 (number of ms)
- Undefined, Null: no change

```
var $o : Object
$o:=New object("a";2)
$o.a:=$o.b // $o.a=0
```

- Assigning an undefined value to a non existing object property does nothing.

When expressions of a given type are expected in your 4D code, you can make sure they have the correct type even when evaluated to undefined by surrounding them with the appropriate 4D cast command: `String`, `Num`, `Date`, `Time`, `Bool`. These commands return an empty value of the specified type when the expression evaluates to undefined. For example:

```
$myString:=Lowercase(String($o.a.b)) //make sure you get a string  
value even if undefined  
//to avoid errors in the code
```

For more information, please refer to [Null and Undefined](#)

## Examples

Using object notation simplifies the 4D code while handling objects. Note however that the command-based notation is still fully supported.

- Writing and reading objects (this example compares object notation and command notation):

```
// Using the object notation  
var $myObj : Object //declares a 4D variable object  
$myObj:={} //creates an object literal and assigns it to the variable  
$myObj.age:=56  
$age:=$myObj.age //56  
  
// Using the command notation  
var $myObj2 : Object //declares a 4D variable object  
OB SET($myObj2;"age";42) //creates an object and adds the age property  
$age:=OB Get($myObj2;"age") //42  
  
// Of course, both notations can be mixed  
var $myObj3 : Object  
OB SET($myObj3;"age";10)  
$age:=$myObj3.age //10
```

- Create a property and assign values, including objects:

```
var $Emp : Object  
$Emp:=New object  
$Emp.city:="London" //creates the city property and sets its value to  
"London"  
$Emp.city:="Paris" //modifies the city property  
$Emp.phone:=New object("office";"123456789";"home";"0011223344")  
//creates the phone property and sets its value to an object
```

- Get a value in a sub-object is very simple using the object notation:

```
$vCity:=$Emp.city //"Paris"  
$vPhone:=$Emp.phone.home //"0011223344"
```

- You can access properties as strings using the [ ] operator

```
$Emp["city"]:="Berlin" //modifies the city property  
//this can be useful for creating properties through variables  
var $addr : Text  
$addr:="address"  
For($i;1;4)  
    $Emp[$addr+String($i)]:=""  
End for  
// creates 4 empty properties "address1...address4" in the $Emp  
object
```



## Picture

A Picture field, variable or expression can be any Windows or Macintosh picture. In general, this includes any picture that can be put on the pasteboard or read from the disk using 4D commands such as `READ PICTURE FILE`.

4D uses native APIs to encode (write) and decode (read) picture fields and variables under both Windows and macOS. These implementations provide access to numerous native formats, including the RAW format, currently used by digital cameras.

- on Windows, 4D uses WIC (Windows Imaging Component).
- on macOS, 4D uses ImageIO.

WIC and ImageIO permit the use of metadata in pictures. Two commands, `SET PICTURE METADATA` and `GET PICTURE METADATA`, let you benefit from metadata in your developments.

## Picture Codec IDs

4D supports natively a wide set of [picture formats](#), such as .jpeg, .png, or .svg.

Picture formats recognized by 4D are returned by the `PICTURE CODEC LIST` command as picture Codec IDs. They can be returned in the following forms:

- As an extension (for example ".gif")
- As a MIME type (for example "image/jpeg")

The form returned for each format will depend on the way the Codec is recorded at the operating system level. Note that the list of available codecs for reading and writing can be different since encoding codecs may require specific licenses.

Most of the [4D picture management commands](#) can receive a Codec ID as a parameter. It is therefore imperative to use the system ID returned by the `PICTURE CODEC LIST` command. Picture formats recognized by 4D are returned by the `PICTURE CODEC LIST` command.

## Picture operators

| <b>Operation</b>          | <b>Syntax Returns</b>        | <b>Action</b>  |
|---------------------------|------------------------------|--|
| Horizontal concatenation  | Pict1 + Pict2 Picture        | Add Pict2 to the right of Pict1  |
| Vertical concatenation    | Pict1 / Pict2 Picture        | Add Pict2 to the bottom of Pict1   |
| Exclusive superimposition | Pict1 & Pict2 Picture        | Superimposes Pict2 on top of Pict1 (Pict2 in foreground). Produces the same result as <code>COMBINE PICTURES(pict3;pict1;Superimposition;pict2)</code>                         |
| Inclusive superimposition | Pict1   Pict2 Picture        | Superimposes Pict2 on Pict1 and returns resulting mask if both pictures are the same size. Produces the same result as <code>\$equal:=Equal pictures(Pict1;Pict2;Pict3)</code> |
| Horizontal move           | Picture<br>+ Number Picture  | Move Picture horizontally Number pixels  |
| Vertical move             | / Number Picture             | Move Picture vertically Number pixels  |
| Resizing                  | * Number Picture             | Resize Picture by Number ratio   |
| Horizontal scaling        | Picture<br>*+ Number Picture | Resize Picture horizontally by Number ratio  |
| Vertical scaling          | *  Number Picture            | Resize Picture vertically by Number ratio  |

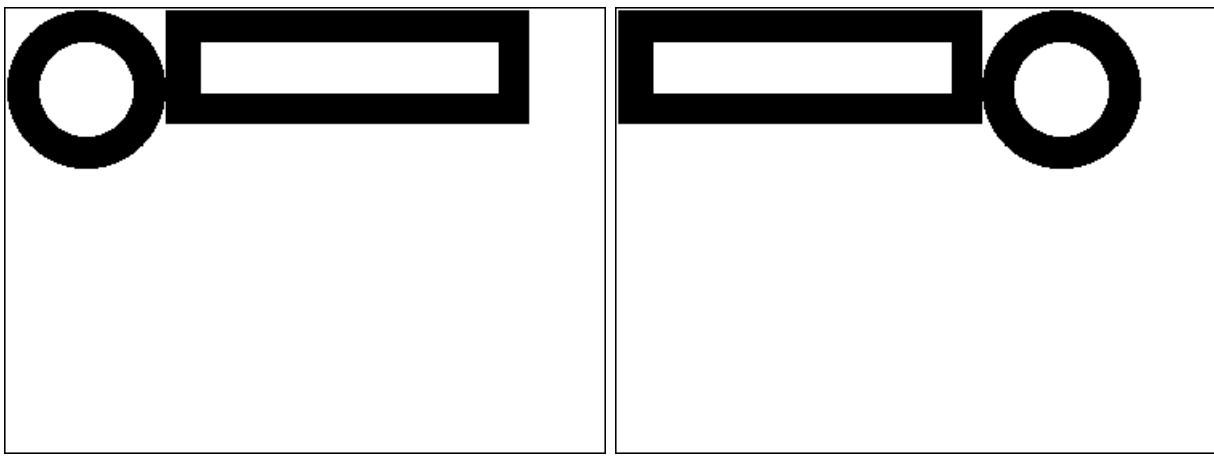
## Notes :

- In order to use the | operator, Pict1 and Pict2 must have exactly the same dimension. If both pictures are a different size, the operation Pict1 | Pict2 produces a blank picture.
- The `COMBINE PICTURES` command can be used to superimpose pictures while keeping the characteristics of each source picture in the resulting picture.
- Additional operations can be performed on pictures using the `TRANSFORM PICTURE` command.
- There is no comparison operators on pictures, however 4D proposes the `Equal picture` command to compare two pictures.

## Examples

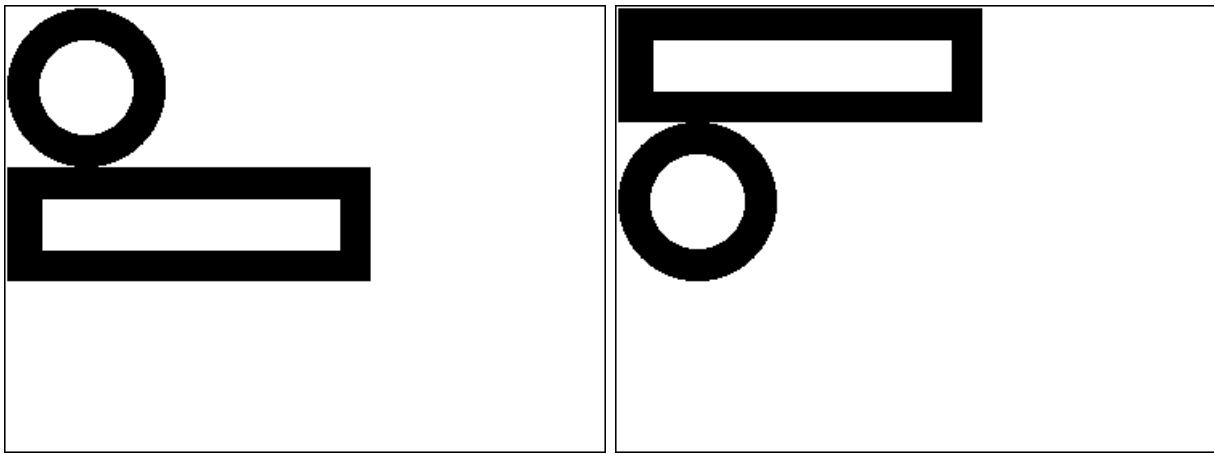
Horizontal concatenation

```
circle+rectangle //Place the rectangle to the right of the circle
rectangle+circle //Place the circle to the right of the rectangle
```



### Vertical concatenation

```
circle/rectangle //Place the rectangle under the circle  
rectangle/circle //Place the circle under the rectangle
```



### Exclusive superimposition

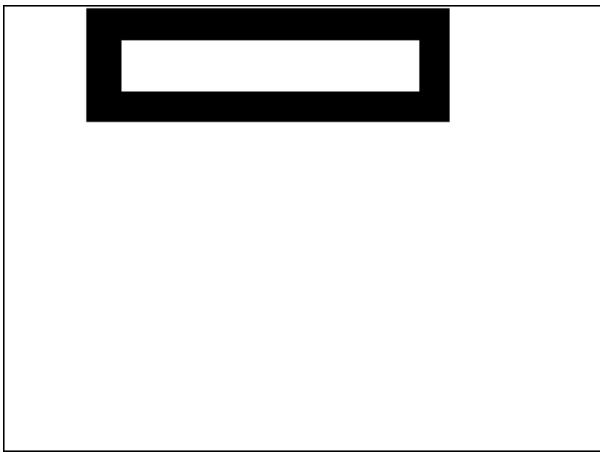
```
Pict3:=Pict1 & Pict2 // Superimposes Pict2 on top of Pict1
```

### Inclusive superimposition

```
Pict3:=Pict1|Pict2 // Recovers resulting mask from superimposing two  
pictures of the same size
```

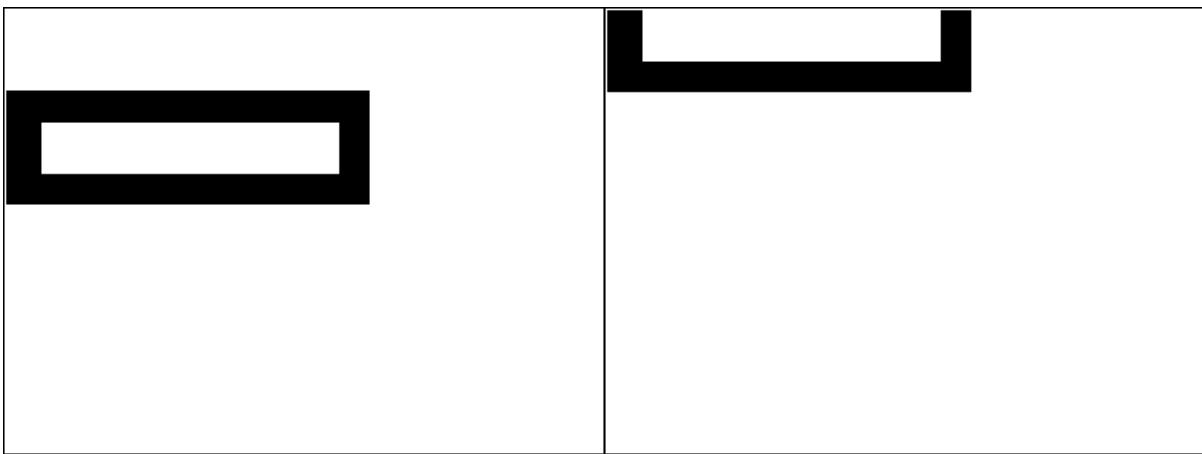
### Horizontal move

```
rectangle+50 //Move the rectangle 50 pixels to the right  
rectangle-50 //Move the rectangle 50 pixels to the left
```



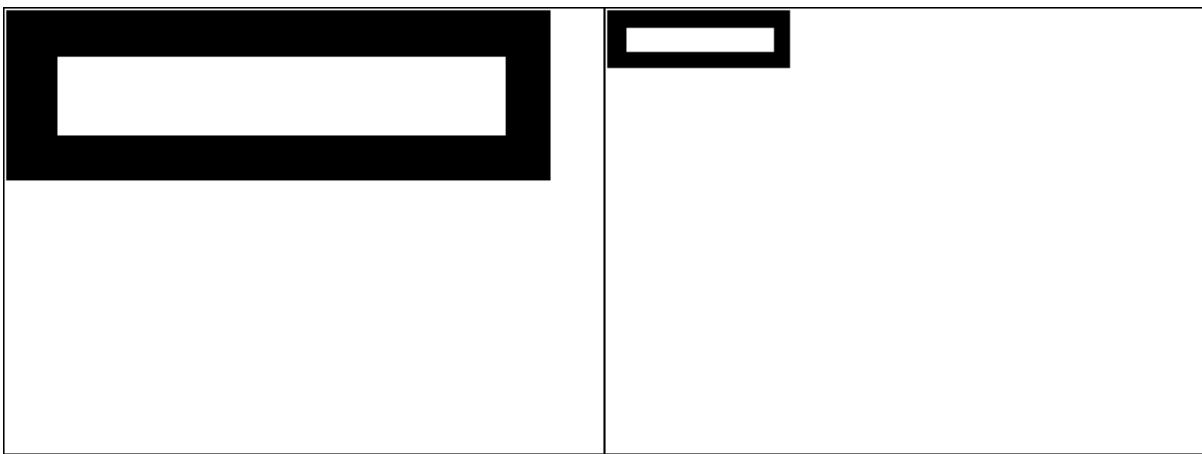
### Vertical move

```
rectangle/50 //Move the rectangle down by 50 pixels  
rectangle/-20 //Move the rectangle up by 20 pixels
```



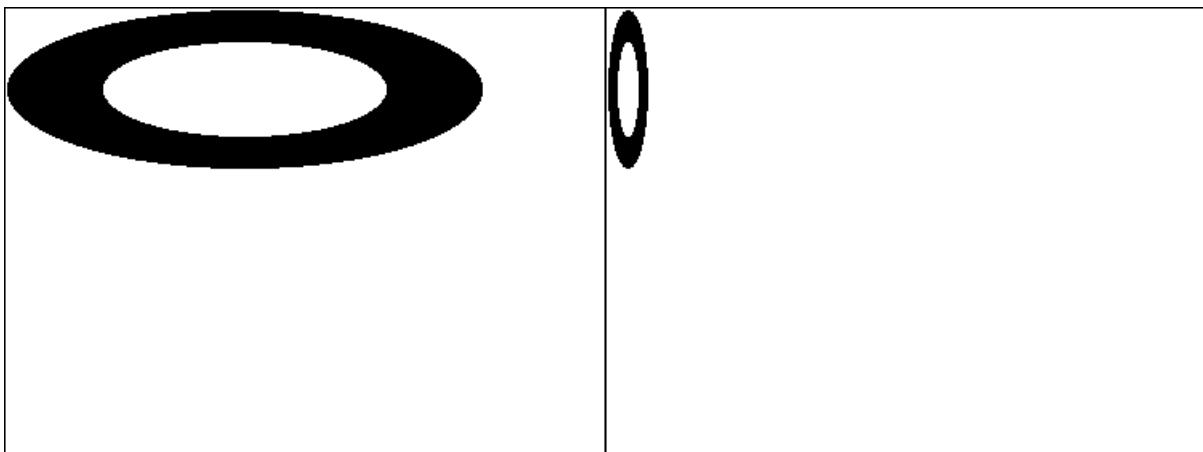
### Resize

```
rectangle*1.5 //The rectangle becomes 50% bigger  
rectangle*0.5 //The rectangle becomes 50% smaller
```



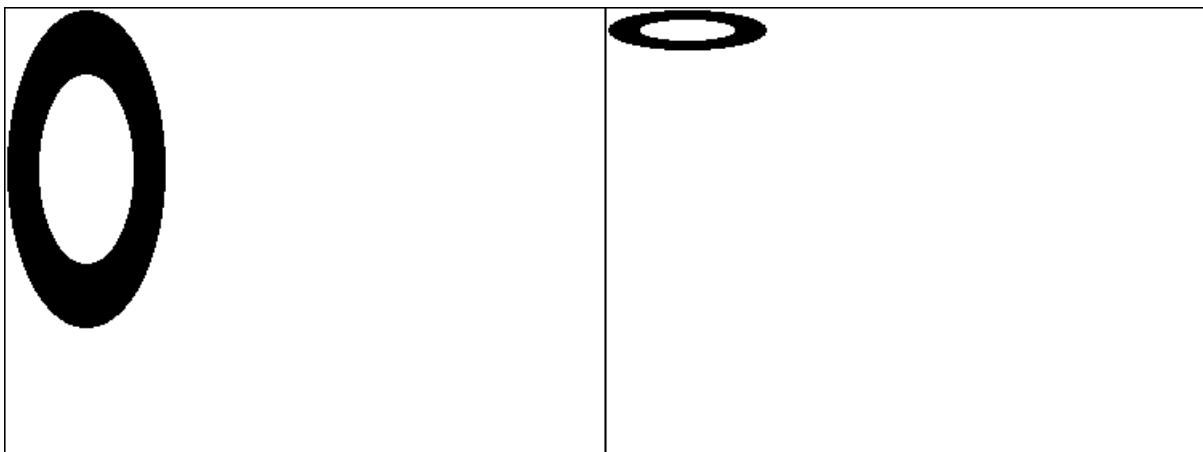
### Horizontal scaling

```
circle*+3 //The circle becomes 3 times wider  
circle*+0.25 //The circle's width becomes a quarter of what it was
```



### Vertical scaling

```
circle*|2 //The circle becomes twice as tall  
circle*|0.25 //The circle's height becomes a quarter of what it was
```



## Pointer

A Pointer variable or expression is a reference to another variable (including arrays and array elements), table, field, or object. There is no field of type Pointer.

Pointers provide an advanced way (in programming) to refer to data. When you use the language, you access various objects—in particular, tables, fields, variables, objects, and arrays—by simply using their names. However, it is often useful to refer to these elements and access them without knowing their names. This is what pointers let you do.

The concept behind pointers is not that uncommon in everyday life. You often refer to something without knowing its exact identity. For example, you might say to a friend, “Let’s go for a ride in your car” instead of “Let’s go for a ride in the car with license plate 123ABD.” In this case, you are referencing the car with license plate 123ABD by using the phrase “your car.” The phrase “car with license plate 123ABD” is like the name of an object, and using the phrase “your car” is like using a pointer to reference the object.

Being able to refer to something without knowing its exact identity is very useful. In fact, your friend could get a new car, and the phrase “your car” would still be accurate—it would still be a car and you could still take a ride in it. Pointers work the same way. For example, a pointer could at one time refer to a numeric field called Age, and later refer to a numeric variable called Old Age. In both cases, the pointer references numeric data that could be used in a calculation.

You can use pointers to reference tables, fields, variables, arrays, array elements, and objects. The following table gives an example of each data type:

| Type          | To Reference            | To Use                   | To Assign                |
|---------------|-------------------------|--------------------------|--------------------------|
| Table         | vpTable:=->[Table]      | DEFAULT TABLE(vpTable->) | n/a                      |
| Field         | vpField:=->[Table]Field | ALERT(vpField->)         | vpField->:="John"        |
| Variable      | vpVar:=->Variable       | ALERT(vpVar->)           | vpVar->:="John"          |
| Array         | vpArr:=->Array          | SORT ARRAY(vpArr->,>)    | COPY ARRAY (Arr;vpArr->) |
| Array element | vpElem:=->Array{1}      | ALERT (vpElem->)         | vpElem->:="John"         |
| Object        | vpObj:=->myObject       | ALERT (vpObj->myProp)    | vpObj->myProp:="John"    |

## Using a pointer: Basic example

It is easiest to explain the use of pointers through an example. This example shows how to access a variable through a pointer. We start by creating a variable:

```
$MyVar := "Hello"
```

\$MyVar is now a variable containing the string “Hello.” We can now create a pointer to \$MyVar:

```
C_POINTER($MyPointer)
$MyPointer:=->$MyVar
```

The -> symbol means “get a pointer to.” This symbol is formed by a dash followed by a “greater than” sign. In this case, it gets the pointer that references or “points to”

`$MyVar`. This pointer is assigned to `MyPointer` with the assignment operator.

`$MyPointer` is now a variable that contains a pointer to `$MyVar`. `$MyPointer` does not contain "Hello", which is the value in `$MyVar`, but you can use `$MyPointer` to get this value. The following expression returns the value in `$MyVar`:

```
$MyPointer->
```

In this case, it returns the string "Hello". The `->` symbol, when it follows a pointer, references the object pointed to. This is called dereferencing.

It is important to understand that you can use a pointer followed by the `->` symbol anywhere that you could have used the object that the pointer points to. This means that you could use the expression `$MyPointer->` anywhere that you could use the original `$MyVar` variable. For example, the following line displays an alert box with the word Hello in it:

```
ALERT ($MyPointer->)
```

You can also use `$MyPointer` to change the data in `$MyVar`. For example, the following statement stores the string "Goodbye" in the variable `$MyVar`:

```
$MyPointer->:="Goodbye"
```

If you examine the two uses of the expression `$MyPointer->`, you will see that it acts just as if you had used `$MyVar` instead. In summary, the following two lines perform the same action—both display an alert box containing the current value in the variable `$MyVar`:

```
ALERT ($MyPointer->)  
ALERT ($MyVar)
```

The following two lines perform the same action— both assign the string "Goodbye" to `$MyVar`:

```
$MyPointer->:="Goodbye"  
$MyVar:="Goodbye"
```

## Pointer operators

With:

```
` vPtrA and vPtrB point to the same object  
vPtrA:==>anObject  
vPtrB:==>anObject  
` vPtrC points to another object  
vPtrC:==>anotherObject
```

| Operation  | Syntax            | Returns Expression Value                                       |
|------------|-------------------|--|
| Equality   | Pointer = Pointer | Boolean<br>vPtrA = vPtrB      True<br>vPtrA = vPtrC      False |
|            |                   |  |
| Inequality | Pointer # Pointer | Boolean<br>vPtrA # vPtrC      True<br>vPtrA # vPtrB      False |
|            |                   |  |

## Main usages

### Pointers to tables

Anywhere that the language expects to see a table, you can use a dereferenced pointer to the table. You create a pointer to a table by using a line like this:

```
$TablePtr:==>[anyTable]
```

You can also get a pointer to a table by using the `Table` command:

```
$TablePtr:=Table(20)
```

You can use the dereferenced pointer in commands, like this:

```
DEFAULT TABLE ($TablePtr->)
```

## Pointers to fields

Anywhere that the language expects to see a field, you can use a dereferenced pointer to reference the field. You create a pointer to a field by using a line like this:

```
$FieldPtr:==>[aTable]ThisField
```

You can also get a pointer to a field by using the `Field` command, for example:

```
$FieldPtr:=Field(1;2)
```

You can use the dereferenced pointer in commands, like this:

```
OBJECT SET FONT ($FieldPtr->;"Arial")
```

## Pointers to local variables

When you use pointers to process or local variables, you must be sure that the variable pointed to is already set when the pointer is used. Keep in mind that local variables are deleted when the method that created them has completed its execution and process variables are deleted at the end of the process that created them. When a pointer calls a variable that no longer exists, this causes a syntax error in interpreted mode (variable not defined) but it can generate a more serious error in compiled mode.

Pointers to local variables allow you to save process variables in many cases. Pointers to local variables can only be used within the same process. In the debugger, when you display a pointer to a local variable that has been declared in another method, the original method name is indicated in parentheses, after the pointer. For example, if you write in Method1:

```
$MyVar:="Hello world"  
Method2 (->$MyVar)
```

In Method2, the debugger will display \$1 as follows:

**\$1->\$MyVar (Method1)**

The value of \$1 will be:

**\$MyVar (Method1) "Hello world"**

## Pointers to array elements

You can create a pointer to an array element. For example, the following lines create an array and assign a pointer to the first array element to a variable called \$ElemPtr:

```
ARRAY REAL ($anArray;10) //Create an array  
$ElemPtr:==>$anArray{1} //Create a pointer to the array element
```

You could use the dereferenced pointer to assign a value to the element, like this:

```
$ElemPtr->:=8
```

## Pointers to arrays

You can create a pointer to an array. For example, the following lines create an array and assign a pointer to the array to a variable called \$ArrPtr:

```
ARRAY REAL($anArray;10) //Create an array  
$ArrPtr:=->$anArray //Create a pointer to the array
```

It is important to understand that the pointer points to the array; it does not point to an element of the array. For example, you can use the dereferenced pointer from the preceding lines like this:

```
SORT ARRAY($ArrPtr->;>) //Sort the array
```

If you need to refer to the fourth element in the array by using the pointer, you do this:

```
ArrPtr->{4}:=84
```

## Pointers as parameters to methods

You can pass a pointer as a parameter to a method. Inside the method, you can modify the object referenced by the pointer. For example, the following method, `takeTwo`, takes two parameters that are pointers. It changes the object referenced by the first parameter to uppercase characters, and the object referenced by the second parameter to lowercase characters. Here is the project method:

```
//takeTwo project method  
//$1 - Pointer to a string field or variable. Change this to  
uppercase.  
//$2 - Pointer to a string field or variable. Change this to  
lowercase.  
$1->:=Uppercase($1->)  
$2->:=Lowercase($2->)
```

The following line uses the `takeTwo` method to change a field to uppercase characters and to change a variable to lowercase characters:

```
takeTwo (->[myTable]myField; ->$MyVar)
```

If the field `[myTable]myField` contained the string "jones", it would be changed to the string "JONES". If the variable `$MyVar` contained the string "HELLO", it would be changed to the string "hello".

In the `takeTwo` method, and in fact, whenever you use pointers, it is important that the data type of the object being referenced is correct. In the previous example, the pointers must point to something that contains a string or text.

## Pointers to pointers

If you really like to complicate things, you can use pointers to reference other pointers. Consider this example:

```
$MyVar:="Hello"  
$PointerOne:=->$MyVar  
$PointerTwo:=->$PointerOne  
($PointerTwo->)->:="Goodbye"  
ALERT(($PointerTwo->)->)
```

It displays an alert box with the word "Goodbye" in it.

Here is an explanation of each line of the example:

- `$MyVar:="Hello"` --> This line puts the string "Hello" into the variable \$MyVar.
- `$PointerOne:-->$MyVar` --> \$PointerOne now contains a pointer to \$MyVar.
- `$PointerTwo:-->$PointerOne` --> \$PointerTwo (a new variable) contains a pointer to \$PointerOne, which in turn points to \$MyVar.
- `($PointerTwo->)->:="Goodbye"` --> \$PointerTwo-> references the contents of \$PointerOne, which in turn references \$MyVar. Therefore `($PointerTwo->)->` references the contents of \$MyVar. So in this case, \$MyVar is assigned "Goodbye".
- `ALERT (($PointerTwo->)->)` --> Same thing: \$PointerTwo-> references the contents of \$PointerOne, which in turn references \$MyVar. Therefore `($PointerTwo->)->` references the contents of \$MyVar. So in this case, the alert box displays the contents of \$MyVar.

The following line puts "Hello" into \$MyVar:

```
($PointerTwo->)->:="Hello"
```

The following line gets "Hello" from \$MyVar and puts it into \$NewVar:

```
$NewVar := ($PointerTwo->)->
```

**Important:** Multiple dereferencing requires parentheses.

## **String**

String is a generic term that stands for:

- Text fields or variables: a Text field, variable, or expression may contain from 0 to 2 GB of text.
- Alphanumeric fields: an Alphanumeric field may contain from 0 to 255 characters (limit set when field is defined).

## **String literals**

A string literal is enclosed in double, straight quotation marks ("..."). Here are some examples of string literals:

```
"Add Records"  
"No records found."  
"Invoice"
```

An empty string is specified by two quotation marks with nothing between them ("").

## **Escape sequences**

The following escape sequences can be used within strings:

### **Escape sequence Character replaced**

|                 |                      |
|-----------------|----------------------|
| ¥n              | LF (Line feed)       |
| ¥t              | HT (Tab)             |
| ¥r              | CR (Carriage return) |
| ¥\¥ (Backslash) |                      |
| ¥"              | " (Quotation marks)  |

**Note:** The ¥ (backslash) character is used as a separator in pathnames under Windows. You must therefore use a double backslash ¥¥ in paths when you want to have a backslash in front of a character used in one of the escape sequences recognized by 4D (e.g. "C:¥¥MyDocuments¥¥New.txt").

## **String operators**

| Operation                | Syntax           | Returns | Expression  | Value         |
|--------------------------|------------------|---------|---|---------------|
| Concatenation            | String + String  | String  | "abc" + "def"                                     | "abcdef"      |
| Repetition               | String * Number  | String  | "ab" * 3  | "ababab"      |
| Equality                 | String = String  | Boolean | "abc" = "abc"<br>"abc" = "abd"                    | True<br>False |
| Inequality               | String # String  | Boolean | "abc" # "abd"<br>"abc" # "abc"                    | True<br>False |
| Greater than             | String > String  | Boolean | "abd" > "abc"<br>"abc" > "abc"                    | True<br>False |
| Less than                | String < String  | Boolean | "abc" < "abd"<br>"abc" < "abc"                    | True<br>False |
| Greater than or equal to | String >= String | Boolean | "abd" >= "abc"<br>"abc" >= "abd"                  | True<br>False |
| Less than or equal to    | String <= String | Boolean | "abc" <= "abd"<br>"abd" <= "abc"                  | True<br>False |
| Contains keyword         | String % String  | Boolean | "Alpha Bravo" % "Bravo"<br>"Alpha Bravo" % "ravo" | True<br>False |
|                          | Picture % String | Boolean | Picture_expr % "Mer"                              | True (*)      |

(\*) If the keyword "Mer" is associated with the picture stored in the picture expression (field or variable).

## String comparisons

- Strings are compared on a character-by-character basis (except in the case of searching by [keywords](#), see below).
- When strings are compared, the case of the characters is ignored; thus, "a"="A" returns `TRUE`. To test if the case of two characters is different, compare their character codes. For example, the following expression returns `FALSE`:

```
Character code("A")=Character code("a") // because 65 is not equal to 97
```

- When strings are compared, diacritical characters are taken into account. For example, the following expressions return `TRUE`:

```
"n"="ñ"  
"n"="Ñ"  
"A"="å"  
// and so on
```

**Note:** String comparison takes into account specificities of the language **defined for the 4D data file** (which is not always the same as the language defined for the system).

## Wildcard character (@)

The 4D language supports @ as a wildcard character. This character can be used in any string comparison to match any number of characters. For example, the following expression is `TRUE`:

```
"abcdefghijkl"="abc@"
```

The wildcard character must be used within the second operand (the string on the

right side) in order to match any number of characters. The following expression is `FALSE`, because the `@` is considered only as a one character in the first operand:

```
"abc@""= "abcdefghij"
```

The wildcard means "one or more characters or nothing". The following expressions are `TRUE`:

```
"abcdefghij"="abcdefghij@"
"abcdefghij"="@abcdefghij"
"abcdefghij"="abcd@efghij"
"abcdefghij"="@abcdefghij@"
"abcdefghij"="@abcde@fghij@"
```

On the other hand, whatever the case, a string comparison with two consecutive wildcards will always return `FALSE`. The following expression is `FALSE`:

```
"abcdefghij"="abc@@fg"
```

When the comparison operator is or contains a `<` or `>` symbol, only comparison with a single wildcard located at the end of the operand is supported:

```
"abcd"<="abc@" // Valid comparison
"abcd"<="abc@ef" //Not a valid comparison
```

If you want to execute comparisons or queries using `@` as a character (and not as a wildcard), you need to use the `Character code (At sign)` instruction. Imagine, for example, that you want to know if a string ends with the `@` character. The following expression (if `$vsValue` is not empty) is always `TRUE`:

```
($vsValue[ [Length ($vsValue) ] ] = "@")
```

The following expression will be evaluated correctly:

```
(Character code($vsValue[ [Length ($vsValue) ] ]) #64)
```

**Note:** A 4D option in the Design environment allows you to define how the `@` character is interpreted when it is included in a character string.

## Keywords

Unlike other string comparisons, searching by keywords looks for "words" in "texts": words are considered both individually and as a whole. The `%` operator always returns `False` if the query concerns several words or only part of a word (for example, a syllable). The "words" are character strings surrounded by "separators," which are spaces and punctuation characters and dashes. An apostrophe, like in "Today's", is usually considered as part of the word, but will be ignored in certain cases (see the rules below). Numbers can be searched for because they are evaluated as a whole (including decimal symbols). Other symbols (currency, temperature, and so on) will be ignored.

```
"Alpha Bravo Charlie%"Bravo" // Returns True
"Alpha Bravo Charlie%"vo" // Returns False
"Alpha Bravo Charlie%"Alpha Bravo" // Returns False
"Alpha,Bravo,Charlie%"Alpha" // Returns True
"Software and Computers%"comput@" // Returns True
```

## Notes:

- 4D uses the ICU library for comparing strings (using `<>=#` operators) and detecting keywords. For more information about the rules implemented, please refer to the following address:

[http://www.unicode.org/reports/tr29/#Word\\_Boundaries](http://www.unicode.org/reports/tr29/#Word_Boundaries).

- In the Japanese version, instead of ICU, 4D uses Mecab by default for detecting keywords.

## Character Reference Symbols

The character reference symbols: [[...]]

These symbols are used to refer to a single character within a string. This syntax allows you to individually address the characters of a text variable, string variable, or field.

If the character reference symbols appear on the left side of the assignment operator (:=), a character is assigned to the referenced position in the string. For example, if vsName is not an empty string, the following line sets the first character of vsName to uppercase:

```
If (vsName#"")
    vsName[ [1] ] := Uppercase (vsName[ [1] ])
End if
```

Otherwise, if the character reference symbols appear within an expression, they return the character (to which they refer) as a 1-character string. For example:

```
//The following example tests if the last character of vtText is an At sign "@"
If (vtText#"")
    If (Character code (Substring (vtText; Length (vtText) ; 1)) = At sign)
//...
    End if
End if

//Using the character reference syntax, you would write in a simpler manner:
If (vtText#"")
    If (Character code (vtText[ [Length (vtText) ] ]) = At sign)
// ...
    End if
End if
```

### Advanced note about invalid character reference

When you use the character reference symbols, you must address existing characters in the string in the same way you address existing elements of an array. For example if you address the 20th character of a string variable, this variable MUST contain at least 20 characters.

- Failing to do so, in interpreted mode, does not cause a syntax error.
- Failing to do so, in compiled mode (with no options), may lead to memory corruption, if, for instance, you write a character beyond the end of a string or a text.
- Failing to do so, in compiled mode, causes an error with the option Range Checking On. For example, executing the following code:

```
//Very bad and nasty thing to do, boo!
vsAnyText:=""
vsAnyText[ [1] ] := "A"
```

will trigger the Runtime Error shown here:

## Syntax Error

Error when executing the method "Method1" at line number 2



Invalid character reference.

```
vsAnyText[[1]]:= "A"
```

Details Edit Trace Continue Abort

## Example

The following project method capitalizes the first character of each word of the text received as parameter and returns the resulting capitalized text:

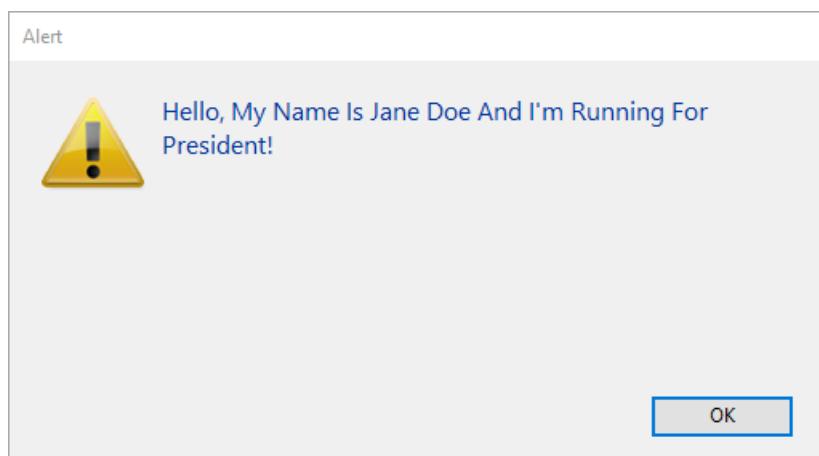
```
//Capitalize_text project method
//Capitalize_text ( Text ) -> Text
//Capitalize_text ( Source text ) -> Capitalized text

$0:=$1
$vlLen:=Length($0)
If ($vlLen>0)
    $0[[1]]:=Uppercase($0[[1]])
    For ($vlChar;1;$vlLen-1)
        If (Position($0[[ $vlChar]]; " !&() -{}:;<>?/, .=+*")>0)
            $0[[ $vlChar+1]]:=Uppercase($0[[ $vlChar+1]])
        End if
    End for
End if
```

For example, the line:

```
ALERT(Capitalize_text("hello, my name is jane doe and i'm running for
president!"))
```

displays the alert shown here:



## Time

A Time field, variable or expression can be in the range of 00:00:00 to 596,000:00:00.

Times are in 24-hour format.

A time value can be treated as a number. The number returned from a time is the number of seconds since midnight (00:00:00) that time represents.

**Note:** In the *4D Language Reference* manual, Time parameters in command descriptions are denoted as Time, except when marked otherwise.

## Time literals

A time literal constant is enclosed by question marks (?...?).

A time literal constant is ordered hour:minute:second, with a colon (:) setting off each part. Times are specified in 24-hour format.

Here are some examples of time literals:

```
?00:00:00? ` midnight  
?09:30:00? ` 9:30 am  
?13:01:59? ` 1 pm, 1 minute, and 59 seconds
```

A null time is specified by ?00:00:00?

**Tip:** The Code Editor includes a shortcut for entering a null time. To type a null time, enter the question mark (?) character and press Enter.

## Time operators

| Operation                | Syntax        | Returns | Expression              | Value      |
|--------------------------|---------------|---------|-------------------------|------------|
| Addition                 | Time + Time   | Time    | ?02:03:04? + ?01:02:03? | ?03:05:07? |
| Subtraction              | Time - Time   | Time    | ?02:03:04? - ?01:02:03? | ?01:01:01? |
| Addition                 | Time + Number | Number  | ?02:03:04? + 65         | 7449       |
| Subtraction              | Time - Number | Number  | ?02:03:04? - 65         | 7319       |
| Multiplication           | Time * Number | Number  | ?02:03:04? * 2          | 14768      |
| Division                 | Time / Number | Number  | ?02:03:04? / 2          | 3692       |
| Longint division         | Time ¥ Number | Number  | ?02:03:04? ¥ 2          | 3692       |
| Modulo                   | Time % Time   | Time    | ?20:10:00? % ?04:20:00? | ?02:50:00? |
| Modulo                   | Time % Number | Number  | ?02:03:04? % 2          | 0          |
| Equality                 | Time = Time   | Boolean | ?01:02:03? = ?01:02:03? | True       |
|                          |               |         | ?01:02:03? = ?01:02:04? | False      |
| Inequality               | Time # Time   | Boolean | ?01:02:03? # ?01:02:04? | True       |
|                          |               |         | ?01:02:03? # ?01:02:03? | False      |
| Greater than             | Time > Time   | Boolean | ?01:02:04? > ?01:02:03? | True       |
|                          |               |         | ?01:02:03? > ?01:02:03? | False      |
| Less than                | Time < Time   | Boolean | ?01:02:03? < ?01:02:04? | True       |
|                          |               |         | ?01:02:03? < ?01:02:03? | False      |
| Greater than or equal to | Time >= Time  | Boolean | ?01:02:03? >=?01:02:03? | True       |
|                          |               |         | ?01:02:03? >=?01:02:04? | False      |
| Less than or equal to    | Time <= Time  | Boolean | ?01:02:03? <=?01:02:03? | True       |
|                          |               |         | ?01:02:04? <=?01:02:03? | False      |

## Example 1

To obtain a time expression from an expression that combines a time expression with a number, use the commands `Time` and `Time string`.

You can combine expressions of the time and number types using the `Time` or `Current time` functions:

```
//The following line assigns to $vlSeconds the number of seconds
//that will be elapsed between midnight and one hour from now
$vlSeconds:=Current time+3600
//The following line assigns to $vhSoon the time it will be in one
hour
$vhSoon:=Time (Current time+3600)
```

The second line could be written in a simpler way:

```
// The following line assigns to $vhSoon the time it will be in one
hour
$vhSoon:=Current time+?01:00:00?
```

## Example 2

The Modulo operator can be used, more specifically, to add times that take the 24-

hour format into account:

```
$t1:=?23:00:00? // It is 23:00 hours  
    // We want to add 2 and a half hours  
$t2:=$t1 +?02:30:00? // With a simple addition, $t2 is ?25:30:00?  
$t2:=($t1 +?02:30:00?)%?24:00:00? // $t2 is ?01:30:00? and it is 1:30  
hour the next morning
```

## Variant

Variant is a variable type which allows encapsulating data of any valid regular type in a variable. Typically, this variable type can be used to write generic code returning or receiving values for which the type is not known. This is the case for example for code handling object attributes.

A variant type variable can contain a value of the following data types:

- BLOB
- boolean
- collection
- date
- longint
- object
- picture
- pointer
- real
- text
- time
- null
- undefined

Arrays cannot be stored in variant variables.

In both interpreted and in compiled modes, a same variant variable can be assigned contents of different types. Unlike regular variable types, the variant variable content type is different from the variant variable type itself. For example:

```
C_VARIANT($variant)

$variant:="hello world"
$vttype:=Type($variant) // 12 (Is variant)
$vttypeVal:=Value type($variant) // 2 (Is text)

$variant:=42
$vttype:=Type($variant) // 12 (Is variant)
$vttypeVal:=Value type($variant) // 1 (Is real)
```

You can use variant variables wherever variables are expected, you only need to make sure than the variable content data type is of the expected type. When accessing variant variables, only their current value is taken into account. For example:

```
C_VARIANT($v)
$v:="hello world"
$v2:=$v //assign variable to another variable

$t:=Type($v) // 12 (Is variant)
$t2:=Type($v2) // 2 (Is text)
```

Variant can be used to declare method parameters (\$0, \$1,...) that can be of various types. In this case, you can build your code by testing the parameter value type, for example:

```
C_VARIANT($1)
Case of
: (Value type($1)=Is longint)
...
: (Value type($1)=Is text)
...
End case
```

When variant variables are not necessary (i.e. when the data type is known), it is recommended to use regular typed variables. Regular typed variables provide better performance, make code more clear and are helpful for the compiler to prevent bugs related to passing unexpected data types.

## Variables

Data in 4D is stored in two fundamentally different ways. **Fields** store data permanently on disk; **variables** store data temporarily in memory.

When you set up your 4D database, you specify the names and types of fields that you want to use. Variables are much the same—you also give them names and different types (see [Data types](#)).

Once created, you can use a variable wherever you need it in your application. For example, you might need to store a text variable in a field of same type:

```
[MyTable]MyField:=MyText
```

Variables are language objects; you can create and use variables that will never appear on the screen. In your forms, you can display variables (except Pointer and BLOB) on the screen, enter data into them, and print them in reports. In this way, enterable and non-enterable area variables act just like fields, and the same built-in controls are available when you create them. Form variables can also control buttons, list boxes, scrollable areas, picture buttons, and so on, or display results of calculations that do not need to be saved.

## Declaring Variables

You create variables by declaring them. The 4D language offers two ways to declare variables:

- using the `var` keyword (recommended, specially if your code uses objects and classes),
- using one of the "Compiler" or "Arrays" theme 4D language commands (classic language only).

**Note:** Although it is usually not recommended, you can create basic variables simply by using them; you do not necessarily need to formally define them. For example, to declare a variable that will hold the current date plus 30 days, you can write:

```
MyDate:=Current date+30 //MyDate is created  
// 4D guesses it is of date type  
// and assigns the current date plus 30 days
```

When variables are declared, they are initialized to the [default value corresponding to their type](#), which they will keep during the session as long as they have not been [assigned](#).

### Using the `var` keyword

Declaring variables using the `var` keyword is recommended since this syntax allows you to bind object variables with classes. Using this syntax enhances code editor suggestions and type-ahead features.

To declare a variable of any type with the `var` keyword, use the following syntax:

```
var <varName>{; <varName2>;...}{ : <varType>}
```

For example:

```

var $myText : Text //a text variable
var myDate1; myDate2 : Date //several date variables
var $myFile : 4D.File //a file class object variable
var $myVar //a variant variable

```

`varName` is the variable name, it must comply with the [4D rules](#) about identifiers. This syntax only supports [local and process variables](#) declarations, thus excluding [interprocess variables](#) and arrays.

`varType` can be:

- a [basic type](#), in which case the variable contains a value of the declared type,
- a [class reference](#) (4D class or user class), in which case the variable contains a reference to an object of the defined class.

If `varType` is omitted, a variable of the **variant** type is created.

The following table lists all supported `varType` values:

| <b>varType</b>            | <b>Contents</b>                                |
|---------------------------|--|
| Text                      | Text value                                     |
| Date                      | Date value                                     |
| Time                      | Time value                                     |
| Boolean                   | Boolean value                                  |
| Integer                   | Long integer value                             |
| Real                      | Real value                                     |
| Pointer                   | Pointer value                                  |
| Picture                   | Picture value                                  |
| Blob                      | Scalar Blob value                              |
| Collection                | Collection value                               |
| Variant                   | Variant value                                  |
| Object                    | Object with default class (4D.Object)          |
| 4D.<className>            | Object of the 4D class name                    |
| cs.<className>            | Object of the user class name                  |
| cs.<namespace><className> | Object of the <namespace> component class name |

## Examples

- To declare local and process basic variables:

```

var $myText; myText; $vt : Text
var myVar //variant

```

```

var $o : Object
//equivalent to:
var $o : 4D.Object
//also equivalent to C_OBJECT($o)

```

- To declare object variables of 4D class:

```

var $myFolder : 4D.Folder
var $myFile : 4D.File

```

- To declare object variables of user class:

```
var $myClass : cs.MyClass  
var $dataclass : cs.Employee  
var $entity : cs.EmployeeEntity
```

## Using a C\_ directive

**Compatibility Note:** This feature is not recommended to declare variables inside methods. It is recommended to use the [var](#) keyword.

Directives from the "Compiler" theme commands allow you to declare variables of basic types.

For example, if you want to define a text variable, you write:

```
C_TEXT(myText)
```

The following are some basic variable declarations:

```
C_BLOB(vxMyBlob) // The process variable vxMyBlob is declared as a  
variable of type BLOB  
C_DATE($vdCurDate) // The local variable $vdCurDate is declared as a  
variable of type Date  
C_LONGINT(vg1;vg2;vg3) // The 3 process variables vg1, vg2 and vg3 are  
declared as variables of type longint  
C_OBJECT($vObj) // The local variable $vObj is declared as a variable  
of type Object  
C_COLLECTION($vCol) // The local variable $vCol is declared as a  
variable of type Collection
```

**Note:** Arrays are a particular type of variables (an array is an ordered series of variables of the same type). Arrays are declared with specific commands, such as `ARRAY LONGINT(alAnArray;10)`. For more information, please refer to [Arrays](#).

## Assigning Data

Data can be put into and copied out of variables and arrays. Putting data into a variable is called **assigning the data to the variable** and is done with the assignment operator (`:=`). The assignment operator is also used to assign data to fields.

The assignment operator is a primary way to create a variable and to put data into it. You write the name of the variable that you want to create on the left side of the assignment operator. For example:

```
MyNumber:=3
```

creates the variable *MyNumber* and puts the number 3 into it. If *MyNumber* already exists, then the number 3 is just put into it.

It is usually not recommended to create variables without [declaring their type](#).

Of course, variables would not be very useful if you could not get data out of them. Once again, you use the assignment operator. If you need to put the value of *MyNumber* in a field called [Products]Size, you would write *MyNumber* on the right side of the assignment operator:

```
[Products]Size:=MyNumber
```

In this case, [Products]Size would be equal to 3. This example is rather simple, but it illustrates the fundamental way that data is transferred from one place to another by using the language.

You assign data to array elements by using curly braces ({...}):

```
atNames{1} := "Richard"
```

## Local, Process, and Interprocess variables

You can create three types of variables: **local**, **process**, and **interprocess**. The difference between the three types of elements is their scope, or the objects to which they are available.

### Local variables

A local variable is, as its name implies, local to a method—accessible only within the method in which it was created and not accessible outside of that method. Being local to a method is formally referred to as being “local in scope.” Local variables are used to restrict a variable so that it works only within the method.

You may want to use a local variable to:

- Avoid conflicts with the names of other variables
- Use data temporarily
- Reduce the number of process variables

The name of a local variable always starts with a dollar sign (\$) and can contain up to 31 additional characters. If you enter a longer name, 4D truncates it to the appropriate length.

When you are working in an application project with many methods and variables, you often find that you need to use a variable only within the method on which you are working. You can create and use a local variable in the method without worrying about whether you have used the same variable name somewhere else.

Frequently, in an application, small pieces of information are needed from the user. The `Request` command can obtain this information. It displays a dialog box with a message prompting the user for a response. When the user enters the response, the command returns the information the user entered. You usually do not need to keep this information in your methods for very long. This is a typical way to use a local variable. Here is an example:

```
$vsID:=Request("Please enter your ID:")
If (OK=1)
    QUERY([People];[People]ID =$vsID)
End if
```

This method simply asks the user to enter an ID. It puts the response into a local variable, \$vsID, and then searches for the ID that the user entered. When this method finishes, the \$vsID local variable is erased from memory. This is fine, because the variable is needed only once and only in this method.

**Note:** Parameters \$1, \$2... passed to methods are local variables. For more information, please refer to [Parameters](#).

### Process variables

A process variable is available only within a process. It is accessible to the process method and any other method called from within the process.

A process variable does not have a prefix before its name. A process variable name can contain up to 31 characters.

In interpreted mode, variables are maintained dynamically; they are created and erased from memory “on the fly.” In compiled mode, all processes you create (user processes) share the same definition of process variables, but each process has a different instance for each variable. For example, the variable myVar is one variable in the process P\_1 and another one in the process P\_2.

A process can “peek and poke” process variables from another process using the commands `GET PROCESS VARIABLE` and `SET PROCESS VARIABLE`. It is good programming practice to restrict the use of these commands to the situation for which they were added to 4D:

- Interprocess communication at specific places or your code
- Handling of interprocess drag and drop
- In Client/Server, communication between processes on client machines and the stored procedures running on the server machines

For more information, see the chapter **Processes** and the description of these commands.

## Interprocess variables

Interprocess variables are available throughout the project and are shared across all cooperative processes. They are primarily used to share information between processes.

Use of interprocess variables is not recommended since they are not available from preemptive processes and tend to make the code less maintainable.

The name of an interprocess variable always begins with the symbols `<>` — a “less than” sign followed by a “greater than” sign— followed by 31 characters.

In Client/Server, each machine (Client machines and Server machine) share the same definition of interprocess variables, but each machine has a different instance for each variable.

## Arrays

An **array** is an ordered series of **variables** of the same type. Each variable is called an **element** of the array. An array is given its size when it is created; you can then resize it as many times as needed by adding, inserting, or deleting elements, or by resizing the array using the same command used to create it. Array elements are numbered from 1 to N, where N is the size of the array. An array always has a special [element zero](#). Arrays are 4D variables. Like any variable, an array has a scope and follows the rules of the 4D language, though with some unique differences.

In most cases, it is recommended to use **collections** instead of **arrays**. Collections are more flexible and provide a wide range of dedicated methods. For more information, please refer to the [Collection](#) section.

## Creating Arrays

You create an array with one of the array declaration commands from the "Array" theme. Each array declaration command can create or resize one-dimensional or two-dimensional arrays. For more information about two-dimensional arrays, see the [two dimensional arrays](#) section.

The following line of code creates (declares) an Integer array of 10 elements:

```
ARRAY INTEGER(aiAnArray;10)
```

Then, the following code resizes that same array to 20 elements:

```
ARRAY INTEGER(aiAnArray;20)
```

Then, the following code resizes that same array to no elements:

```
ARRAY INTEGER(aiAnArray;0)
```

## Assigning values in arrays

You reference the elements in an array by using curly braces (`{...}`). A number is used within the braces to address a particular element; this number is called the element number. The following lines put five names into the array called atNames and then display them in alert windows:

```
ARRAY TEXT(atNames;5)
atNames{1} :="Richard"
atNames{2} :="Sarah"
atNames{3} :="Sam"
atNames{4} :="Jane"
atNames{5} :="John"
For($v1Elem;1;5)
    ALERT("The element #"+String($v1Elem)+" is equal to:
"+atNames{$v1Elem})
End for
```

Note the syntax `atNames{$v1Elem}`. Rather than specifying a numeric literal such as `atNames{3}`, you can use a numeric variable to indicate which element of an array you are addressing. Using the iteration provided by a loop structure ([For...End for](#), [Repeat...Until](#) or [While...End while](#)), compact pieces of code can address all or part of the elements in an array.

**Important:** Be careful not to confuse the assignment operator (`:=`) with the

comparison operator, equal (=). Assignment and comparison are very different operations.

## Assigning an array to another array

Unlike text or string variables, you cannot assign one array to another. To copy (assign) an array to another one, use `COPY ARRAY`.

## Using the element zero of an array

An array always has an element zero. While element zero is not shown when an array supports a form object, there is no restriction(\*) in using it with the language.

Here is another example: you want to initialize a form object with a text value but without setting a default value. You can use the element zero of the array:

```
// method for a combo box or drop-down list
// bound to atName variable array
Case of
  :(Form event code=On Load)
  // Initialize the array (as shown further above)
  // But use the element zero
    ARRAY TEXT(atName;5)
    atName{0}:=Please select an item"
    atName{1}:="Text1"
    atName{2}:="Text2"
    atName{3}:="Text3"
    atName{4}:="Text4"
    atName{5}:="Text5"
  // Position the array to element 0
  atName:=0
End case
```

(\*) However, there is one exception: in an array type List Box, the zero element is used internally to store the previous value of an element being edited, so it is not possible to use it in this particular context.

## Two-dimensional Arrays

Each of the array declaration commands can create or resize one-dimensional or two-dimensional arrays. Example:

```
ARRAY TEXT(atTopics;100;50) // Creates a text array composed of 100
rows of 50 columns
```

Two-dimensional arrays are essentially language objects; you can neither display nor print them.

In the previous example:

- `atTopics` is a two-dimensional array
- `atTopics{8}{5}` is the 5th element (5th column...) of the 8th row
- `atTopics{20}` is the 20th row and is itself a one-dimensional array
- `Size of array(atTopics)` returns 100, which is the number of rows
- `Size of array(atTopics{17})` returns 50, which the number of columns for the 17th row

In the following example, a pointer to each field of each table in the database is stored in a two-dimensional array:

```

C_LONGINT($vlLastTable;$vlLastField)
C_LONGINT($vlFieldNumber)
// Create as many rows (empty and without columns) as there are
tables
$vlLastTable:=Get last table number
ARRAY POINTER(<>apFields;$vlLastTable;0) //2D array with X rows and
zero columns
// For each table
For($vlTable;1;$vlLastTable)
    If(Is table number valid($vlTable))
        $vlLastField:=Get last field number($vlTable)
    // Give value of elements
        $vlColumnNumber:=0
        For($vlField;1;$vlLastField)
            If(Is field number valid($vlTable;$vlField))
                $vlColumnNumber:=$vlColumnNumber+1
        //Insert a column in a row of the table underway
            INSERT IN ARRAY(<>apFields{$vlTable};$vlColumnNumber;1)
        //Assign the "cell" with the pointer
            <>apFields{$vlTable}
{$vlColumnNumber}:=Field($vlTable;$vlField)
        End if
    End for
End if
End for

```

Provided that this two-dimensional array has been initialized, you can obtain the pointers to the fields for a particular table in the following way:

```

// Get the pointers to the fields for the table currently displayed
at the screen:
COPY ARRAY(◊apFields{Table(Current form
table)});$apTheFieldsIamWorkingOn)
// Initialize Boolean and Date fields
For($vlElem;1;Size of array($apTheFieldsIamWorkingOn))
    Case of
        : (Type ($apTheFieldsIamWorkingOn{$vlElem}->) = Is date)
            $apTheFieldsIamWorkingOn{$vlElem}->:=Current date
        : (Type ($apTheFieldsIamWorkingOn{$vlElem}->) = Is Boolean)
            $apTheFieldsIamWorkingOn{$vlElem}->:=True
    End case
End for

```

**Note:** As this example suggests, rows of a two-dimensional arrays can be the same size or different sizes.

## Arrays and Memory

Unlike the data you store on disk using tables and records, an array is always held in memory in its entirety.

For example, if all US zip codes were entered in the [Zip Codes] table, it would contain about 100,000 records. In addition, that table would include several fields: the zip code itself and the corresponding city, county, and state. If you select only the zip codes from California, the 4D database engine creates the corresponding selection of records within the [Zip Codes] table, and then loads the records only when they are needed (i.e., when they are displayed or printed). In other words, you work with an ordered series of values (of the same type for each field) that is partially loaded from the disk into the memory by the database engine of 4D.

Doing the same thing with arrays would be prohibitive for the following reasons:

- In order to maintain the four information types (zip code, city, county, state), you

would have to maintain four large arrays in memory.

- Because an array is always held in memory in its entirety, you would have to keep all the zip codes information in memory throughout the whole working session, even though the data is not always in use.
- Again, because an array is always held in memory in its entirety, each time the application is started and then quit, the four arrays would have to be loaded and then saved on the disk, even though the data is not used or modified during the working session.

**Conclusion:** Arrays are intended to hold reasonable amounts of data for a short period of time. On the other hand, because arrays are held in memory, they are easy to handle and quick to manipulate.

However, in some circumstances, you may need to work with arrays holding hundreds or thousands of elements. The following table lists the formulas used to calculate the amount of memory used for each array type:

| <b>Array Type</b> | <b>Formula for determining Memory Usage in Bytes</b>                       |
|-------------------|--|
| Blob              | (1+number of elements) * 12 + Sum of the size of each blob                 |
| Boolean           | (31+number of elements)*8  |
| Date              | (1+number of elements) * 6   |
| Integer           | (1+number of elements) * 2   |
| Long Integer      | (1+number of elements) * 4   |
| Object            | (1+number of elements) * 8 + Sum of the size of each object                |
| Picture           | (1+number of elements) * 8 + Sum of the size of each picture               |
| Pointer           | (1+number of elements) * 8 + Sum of the size of each pointer               |
| Real              | (1+number of elements) * 8   |
| Text              | (1+number of elements) 20 + ( <i>Sum of the length of each text</i> )<br>2 |
| Time              | (1+number of elements) * 4   |
| Two-dimensional   | (1+number of elements) * 16 + Sum of the size of each array                |

#### **Notes:**

- The size of a text in memory is calculated using this formula: ((Length + 1) \* 2)
- A few additional bytes are required to keep track of the selected element, the number of elements, and the array itself.

## Methods

A method is basically a piece of code that executes one or several action(s). A method is composed of statements.

A statement performs an action, and may be simple or complex. Each statement usually consists of one line in the method (if necessary, it can however be [split using the \ character](#)).

The maximum size of a method is limited to 2 GB of text or 32,000 lines of code.

## Method Types

In the 4D Language, there are several categories of methods. The category depends on how they can be called:

| Type                       | Calling context  | Accepts parameters | Description  |
|----------------------------|--|--------------------|--|
| Project method             | On demand, when the project method name is called (see <a href="#">Calling project methods</a> ) | Yes                | Can contain any code to execute any custom actions. Once a project method is created, it becomes part of the language of the project.                                |
| Object (widget) method     | Automatic, when an event involves the object to which the method is attached                     | No                 | Property of a form object (also called widget)   |
| Form method                | Automatic, when an event involves the form to which the method is attached                       | No                 | Property of a form. You can use a form method to manage data and objects, but it is generally simpler and more efficient to use an object method for these purposes. |
| Trigger (aka Table method) | Automatic, each time that you manipulate the records of a table (Add, Delete and Modify)         | No                 | Property of a table. Triggers are methods that can prevent "illegal" operations with the records of your database.   |
| Database method            | Automatic, when a working session event occurs   | Yes (predefined)   | There are 16 database methods in 4D.   |
| Class                      | <a href="#">Class functions</a> are called in the context of an object instance                  | yes                | Class functions can be built-in (e.g. <code>collection.orderBy()</code> or <code>entity.save()</code> ), or created by the 4D developer. See <a href="#">Classes</a> |

## Calling Project Methods

A project method can have one of the following roles, depending on how it is executed and used:

- Subroutine
- Object formula
- Menu method
- Process method
- Event or Error catching method

You can also execute your project methods manually, for testing purpose for example.

## Subroutines

A subroutine is a project method that can be thought of as a servant. It performs those tasks that other methods request it to perform. A function is a subroutine that returns a value to the method that called it.

When you create a project method, it becomes part of the language of the project in which you create it. You can then call the project method from another method (project method, object method...) in the same way that you call 4D's built-in commands. A project method used in this way is called a subroutine.

You use subroutines to:

- Reduce repetitive coding
- Clarify your methods
- Facilitate changes to your methods
- Modularize your code

For example, let's say you have a project of customers. As you customize the project, you find that there are some tasks that you perform repeatedly, such as finding a customer and modifying his or her record. The code to do this might look like this:

```
// Look for a customer
QUERY BY EXAMPLE([Customers])
// Select the input form
FORM SET INPUT([Customers];"Data Entry")
// Modify the customer's record
MODIFY RECORD([Customers])
```

If you do not use subroutines, you will have to write the code each time you want to modify a customer's record. If there are ten places in your project where you need to do this, you will have to write the code ten times. If you use subroutines, you will only have to write it once. This is the first advantage of subroutines—to reduce the amount of code.

If the previously described code was a method called `MODIFY_CUSTOMER`, you would execute it simply by using the name of the method in another method. For example, to modify a customer's record and then print the record, you would write this method:

```
MODIFY_CUSTOMER
PRINT SELECTION([Customers])
```

This capability simplifies your methods dramatically. In the example, you do not need to know how the `MODIFY_CUSTOMER` method works, just what it does. This is the second reason for using subroutines—to clarify your methods. In this way, your methods become extensions to the 4D language.

If you need to change your method of finding customers in this example project, you will need to change only one method, not ten. This is the next reason to use subroutines—to facilitate changes to your methods.

Using subroutines, you make your code modular. This simply means dividing your code into modules (subroutines), each of which performs a logical task. Consider the following code from a checking account project:

```
FIND_CLEARED_CHECKS //Find the cleared checks
RECONCILE_ACCOUNT //Reconcile the account
PRINT_CHECK_BOOK_REPORT //Print a checkbook report
```

Even for someone who doesn't know the project, it is clear what this code does. It is not necessary to examine each subroutine. Each subroutine might be many lines long and perform some complex operations, but here it is only important that it performs its task. We recommend that you divide your code into logical tasks, or modules, whenever possible.

## Object formulas

You can encapsulate your project methods in **formula** objects and call them from your objects.

The `Formula` or `Formula from string` commands allow you to create native formula objects that you can encapsulate in object properties. It allows you to implement custom object methods.

To execute a method stored in an object property, use the `()` operator after the property name. For example:

```
//myAlert  
ALERT("Hello world!")
```

Then `myAlert` can be encapsulated in any object and called:

```
var $o : Object  
$o:=New object("custom_Alert";Formula(myAlert))  
$o.custom_Alert() //displays "Hello world!"
```

Syntax with brackets is also supported:

```
$o["custom_Alert"]() //displays "Hello world!"
```

You can also [pass parameters](#) to your formula when you call it by using `$1, $2...` just like with 4D project methods:

```
//fullName method  
C_TEXT($0;$1;$2)  
$0:=$1+" "+$2
```

You can encapsulate `fullName` in an object:

```
var $o : Object  
$o:=New object("full_name";Formula(fullName))  
$result:=$o.full_name("John";"Smith")  
//$result = "John Smith"  
//equivalent to $result:=fullName("param1";"param2")
```

Combined with the `This` function, such object methods allow writing powerful generic code. For example:

```
//fullName2 method  
C_TEXT($0)  
$0:=This.firstName+" "+This.lastName
```

Then the method acts like a new, calculated attribute that can be added to other attributes:

```
var $o : Object  
$o:=New object("firstName";"Jim";"lastName";"Wesson")  
$o.fullName:=Formula(fullName2) //add the method  
  
$result:=$o.fullName()  
//$result = "Jim Wesson"
```

Note that, even if it does not have parameters, an object method to be executed must be called with `()` parenthesis. Calling only the object property will return a new reference to the formula (and will not execute it):

```
$o:=$f.message //returns the formula object in $o
```

## Menu Methods

A menu method is invoked when you select the custom menu command to which it is attached. You assign the method to the menu command using the Menu editor or a command of the "Menus" theme. The method executes when the menu command is chosen. By creating custom menus with menu methods that perform specific actions, you create custom interfaces for your desktop applications.

Custom menu commands can cause one or more activities to take place. For example, a menu command for entering records might call a method that performs two tasks: displaying the appropriate input form, and calling the `ADD RECORD` command until the user cancels the data entry activity.

Automating sequences of activities is a very powerful capability of the programming language. Using custom menus, you can automate task sequences and thus provide more guidance to users of the application.

## Process Methods

A **process method** is a project method that is called when a process is started. The process lasts only as long as the process method continues to execute, except if it is a Worker process. Note that a menu method attached to a menu command with *Start a New Process* property is also the process method for the newly started process.

## Event and Error catching Methods

An **event catching method** runs in a separate process as the process method for catching events. Usually, you let 4D do most of the event handling for you. For example, during data entry, 4D detects keystrokes and clicks, then calls the correct object and form methods so you can respond appropriately to the events from within these methods. For more information, see the description of the command `ON EVENT CALL`.

An **error catching method** is an interrupt-based project method. It is called each time an error or an exception occurs. For more information, see the [Error handling](#) section.

## Manual Execution

Project methods written in your application are usually called automatically during the use of the application via menu commands, buttons, other methods, and so on. As for database methods, they are executed in relation to specific events that occur in the application.

However, for testing and debugging purposes, 4D lets you manually execute project methods and certain database methods in Design mode. In this case, it is possible to run the method in a new process and/or directly in Debug mode, in order to check its execution step by step.

You can execute methods in two ways:

- From the Code Editor window,
- From the Execute Method dialog box (project methods only).

## From the Code Editor

Each **Code Editor** window has a button that can be used to run the current method. Using the menu associated with this button, you can choose the type of execution desired.

This button is only active for project methods and for the following database methods:

- On Startup
- On Exit
- On Server Startup
- On Server Shutdown

For more information, see [Toolbar](#).

## From the Execute Method dialog box

When you select the **Method...** command of the **Run** menu, displays the **Execute Method** dialog.

This dialog box lists all the project methods of the database, including shared project methods of components. On the other hand, project methods that have been declared invisible will not appear.

To execute a project method, simply select its name in the list and click on **Execute**. To run a method step by step in Debug mode, click on **Debug**. For more information about the 4D debugger, refer to the [Debugging](#) section.

If you check the **New Process** check box, the method you selected executes in another process. If the method is performing a time-consuming task such as printing a large set of records, you can continue to work with your database, adding records to a table, creating a graph to display data, and so on. For more information about processes, refer to [Processes](#) the 4D Language Reference manual.

### 4D Server Notes:

- If you want the method to be executed on the server machine rather than on the client machine, select the **On 4D Server** option in the To be executed menu. In this case, a new process, call a *stored procedure*, is created on the server machine in order to execute the method. This option can be used to reduce network traffic and optimize the functioning of 4D Server, in particular for methods that call data stored on the disk. All types of methods can be executed on the server machine or on another client machine, except for those that modify the user interface. In this case, stored procedures are ineffective.
- You can also choose to run the method on another client workstation. Other client workstations will not appear in the menu, unless they have been previously "registered" (for more information, refer to the description of the [REGISTER CLIENT](#)).

By default, the **locally** option is selected. With the 4D single-user version, this is the only option available.

## Recursive Project Methods

Project methods can call themselves. For example:

- The method A may call the method B which may call A, so A will call B again and so on.
- A method can call itself.

This is called recursion. The 4D language fully supports recursion.

Here is an example. Let's say you have a [Friends and Relatives] table composed of this extremely simplified set of fields:

- [Friends and Relatives]Name
- [Friends and Relatives]ChildrensName

For this example, we assume the values in the fields are unique (there are no two persons with the same name). Given a name, you want to build the sentence "A friend of mine, John who is the child of Paul who is the child of Jane who is the child of Robert who is the child of Eleanor, does this for a living!":

1. You can build the sentence in this way:

```
$vsName:=Request("Enter the name:","John")
If(OK=1)
    QUERY([Friends and Relatives];[Friends and Relatives]Name=$vsName)
    If(Records in selection([Friends and Relatives])>0)
        $vtTheWholeStory:="A friend of mine, "+$vsName
        Repeat
            QUERY([Friends and Relatives];[Friends and
Relatives]ChildrensName=$vsName)
            $vlQueryResult:=Records in selection([Friends and
Relatives])
            If($vlQueryResult>0)
                $vtTheWholeStory:=$vtTheWholeStory+" who is the child of
"+[Friends and Relatives]Name
                $vsName:=[Friends and Relatives]Name
            End if
            Until($vlQueryResult=0)
            $vtTheWholeStory:=$vtTheWholeStory+", does this for a living!"
            ALERT($vtTheWholeStory)
        End if
    End if
```

2. You can also build it this way:

```
$vsName:=Request("Enter the name:","John")
If(OK=1)
    QUERY([Friends and Relatives];[Friends and Relatives]Name=$vsName)
    If(Records in selection([Friends and Relatives])>0)
        ALERT("A friend of mine, "+Genealogy of($vsName)+" , does this
for a living!")
    End if
End if
```

with the recursive function Genealogy of listed here:

```
` Genealogy of project method
` Genealogy of ( String ) -> Text
` Genealogy of ( Name ) -> Part of sentence

$0:=$1
QUERY([Friends and Relatives];[Friends and Relatives]ChildrensName=$1)
If(Records in selection([Friends and Relatives])>0)
    $0:=$0+" who is the child of "+Genealogy of([Friends and
Relatives]Name)
End if
```

Note the Genealogy of method which calls itself.

The first way is an **iterative algorithm**. The second way is a **recursive algorithm**.

When implementing code for cases like the previous example, it is important to note that you can always write methods using iteration or recursion. Typically, recursion provides more concise, readable, and maintainable code, but using it is not mandatory.

Some typical uses of recursion in 4D are:

- Treating records within tables that relate to each other in the same way as in the example.
- Browsing documents and folders on your disk, using the commands `FOLDER LIST` and `DOCUMENT LIST`. A folder may contain folders and documents, the subfolders can themselves contain folders and documents, and so on.

**Important:** Recursive calls should always end at some point. In the example, the method `Genealogy of` stops calling itself when the query returns no records. Without this condition test, the method would call itself indefinitely; eventually, 4D would return a "Stack Full" error because it would no longer have space to "pile up" the calls (as well as parameters and local variables used in the method).

## Parameters

You'll often find that you need to pass data to your methods and functions. This is easily done with parameters.

### Overview

**Parameters** (or **arguments**) are pieces of data that a method or a class function needs in order to perform its task. The terms *parameter* and *argument* are used interchangeably throughout this manual. Parameters are also passed to built-in 4D commands. In this example, the string "Hello" is an argument to the `ALERT` built-in command:

```
ALERT("Hello")
```

Parameters are passed to methods or class functions in the same way. For example, if a class function named `getArea()` accepts two parameters, a call to the class function might look like this:

```
$area:=$o.getArea(50;100)
```

Or, if a project method named `DO_SOMETHING` accepts three parameters, a call to the method might look like this:

```
DO_SOMETHING($WithThis;$AndThat;$ThisWay)
```

The input parameters are separated by semicolons (;).

The same principles are used when methods are executed through dedicated commands, for example:

```
EXECUTE METHOD IN SUBFORM("Cal2";"SetCalendarDate";*;!05/05/20!)
//pass the !05/05/20! date as parameter to the SetCalendarDate
//in the context of a subform
```

Data can also be **returned** from methods and class functions. For example, the following line is a statement that uses the built-in command, `Length`, to return the length of a string. The statement puts the value returned by `Length` in a variable called `MyLength`. Here is the statement:

```
MyLength:=Length("How did I get here?")
```

Any subroutine can return a value. Only one single output parameter can be declared per method or class function.

Input and output values are [evaluated](#) at the moment of the call and copied into or from local variables within the called class function or method. Variable parameters must be [declared](#) in the called code.

#### COMPATIBILITY

Throughout the 4D documentation, you might see examples where parameters are automatically copied in sequentially numbered local variables (\$0, \$1, etc.) and declared using compiler directives. Ex: `C_TEXT($1;$2)`. This legacy syntax is still supported but is no longer recommended.

## Declaring parameters

Inside called methods or class functions, parameter values are assigned to local variables. You usually declare parameters using a **parameter name** along with a **parameter type**, separated by colon.

- For class functions, parameters are declared along with the `Function` keyword.
- For methods (project methods, form object methods, database methods, and triggers), parameters are declared using the `#DECLARE` keyword at the beginning of the method code.

Examples:

```
Function getArea($width : Integer; $height : Integer) -> $area : Integer
  //myProjectMethod
#DECLARE ($i : Integer) -> $myResult : Object
```

The following rules apply:

- The declaration line must be the first line of the method or function code, otherwise an error is displayed (only comments or line breaks can precede the declaration).
- Parameter names must start with a `$` character and be compliant with [property naming rules](#).
- Multiple parameters (and types) are separated by semicolons (`;`).
- Multiline syntaxes are supported (using `"$"` character).

For example, when you call a `getArea()` function with two parameters:

```
$area:=$o.getArea(50;100)
```

In the class function code, the value of each parameter is copied into the corresponding declared parameter:

```
// Class: Polygon
Function getArea($width : Integer; $height : Integer) -> $area : Integer
  $area:=$width*$height
```

If the type is not defined, the parameter will be defined as [Variant](#).

All 4D method kinds support the `#DECLARE` keyword, including database methods. For example, in the `On Web Authentication` database method, you can declare named parameters:

```
// On Web Authentication database method
#DECLARE ($url : Text; $header : Text; \
  $BrowserIP : Text; $ServerIP : Text; \
  $user : Text; $password : Text) \
-> $RequestAccepted : Boolean
$entitySelection:=ds.User.query("login=:1"; $user)
// Check hash password...
```

## Returned value

You declare the return parameter of a function by adding an arrow (`->`) and the parameter definition after the input parameter(s) list. For example:

```
Function add($x : Variant; $y : Integer) -> $result : Integer
```

You can also declare the return parameter only by adding `: type`, in which case it can be handled by a [return statement](#). For example:

```
Function add($x : Variant; $y : Integer) : Integer
    return $x+$y
```

## Supported data types

With named parameters, you can use the same data types as those which are [supported by the `var` keyword](#), including class objects. For example:

```
Function saveToFile($entity : cs.ShapesEntity; $file : 4D.File)
```



**NOTE**

Tables or array expressions can only be passed [as reference using a pointer](#).

## Initialization

When parameters are declared, they are initialized to the [default value corresponding to their type](#), which they will keep during the session as long as they have not been assigned.

```
return {expression}
```

- History

The `return` statement ends function or method execution and can be used to return an expression to the caller.

For example, the following function returns the square of its argument, `$x`, where `$x` is a number.

```
Function square($x : Integer)
    return $x * $x
```

Internally, `return x` executes `$0:=x` or (if declared) `myReturnValue:=x`, and returns to the caller. If `return` is used without an expression, the function or method returns a null value of the declared return type (if any), otherwise *undefined*.

The `return` statement can be used along with the standard syntax for [returned values](#) (the returned value must be of the declared type). However, note that it ends immediately the code execution. For example:

```
Function getValue
    $0:=10
    return 20
    // returns 20
```

```
Function getValue -> $v : Integer
    return 10
    $v:=20 // never executed
    // returns 10
```

## Parameter indirection ( `${N}` )

4D project methods accept a variable number of parameters. You can address those parameters with a `For...End for` loop, the [Count parameters](#) command and the **parameter indirection syntax**. Within the method, an indirection address is formatted  `${N}` , where `N` is a numeric expression.  `${N}`  is called a **generic parameter**.

## Using generic parameters

For example, consider a method that adds values and returns the sum formatted according to a format that is passed as a parameter. Each time this method is called, the number of values to be added may vary. We must pass the values as parameters to the method and the format in the form of a character string. The number of values can vary from call to call.

Here is the method, named `MySum`:

```
#DECLARE($format : Text) -> $result : Text
$sum:=0
For($i;2;Count parameters)
    $sum:=$sum+$i
End for
$result:=String($sum;$format)
```

The method's parameters must be passed in the correct order, first the format and then a variable number of values:

```
Result:=MySum("#0.00";125,2;33,5;24) //"182.70"
Result:=MySum("000";1;2;200) //"203"
```

Note that even if you declared 0, 1, or more parameters in the method, you can always pass the number of parameters that you want. Parameters are all available within the called method through the  `${N}` syntax and extra parameters type is [Variant](#) by default (you can declare them using a [compiler directive](#)). You just need to make sure parameters exist, thanks to the [Count parameters](#) command. For example:

```
//foo method
#DECLARE($p1: Text;$p2 : Text; $p3 : Date)
For($i;1;Count parameters)
    ALERT("param "+String($i)+" = "+String(${i}))
End for
```

This method can be called:

```
foo("hello";"world";!01/01/2021!;42;?12:00:00?) //extra parameters are
passed
```

Parameter indirection is best managed if you respect the following convention: if only some of the parameters are addressed by indirection, they should be passed after the others.

## Declaring generic parameters

As with other local variables, it is not mandatory to declare generic parameters by compiler directive. However, it is recommended to avoid any ambiguity. Non-declared generic parameters automatically get the [Variant](#) type.

To declare generic parameters, you use a compiler directive to which you pass  `${N}` as a parameter, where N specifies the first generic parameter.

```
C_TEXT(${4})
```

This command means that starting with the fourth parameter (included), the method can receive a variable number of parameters of text type. The first three parameters can be of any data type. However, if you use `$2` by indirection, the data type used will be the generic type. Thus, it will be of the data type text, even if for you it was, for instance, of the data type Real.

The number in the declaration has to be a constant and not a variable.

## Compiler method

Even if it is not mandatory in [interpreted mode](#), you must declare each parameter in the called methods or functions as soon as you intend to compile your project.

When using the `#DECLARE` keyword or `Function` prototype, parameters are automatically declared. For example:

```
Function add($x : Variant; $y : Integer) -> $result : Integer  
    // all parameters are declared with their type
```

However, a 4D compiler feature allows you to declare all your parameters in a specific method using a special syntax:

- you can group all local variable parameters for project methods in one or more project method(s)
- the method name(s) must start with "**Compiler**", for example "Compiler\_MyParameters".
- within such a method, you can predeclare the parameters for each method using the following syntax: `C_XXX(methodName;parameter)`.

For example:

```
// Compiler_method  
C_REAL(OneMethodAmongOthers;$myParam)
```

### NOTE

This syntax is not executable in interpreted mode.

You can create and fill automatically a `Compiler` method containing all your parameters using the [Compiler Methods for... Methods](#) button in the Compiler Settings dialog box.

Parameter declaration is also mandatory in the following contexts (these contexts do not support declaration in a "Compiler" method):

- Database methods - For example, the `On Web Connection Database Method` receives six parameters of the data type Text. At the beginning of the database method, you must write (even if all parameters are not used):

```
// On Web Connection  
#DECLARE ($url : Text; $header : Text; \  
$BrowserIP : Text; $ServerIP : Text; \  
$user : Text; $password : Text) \  
-> $RequestAccepted : Boolean
```

- Triggers - The `$0` parameter (Longint), which is the result of a trigger, will be typed by the compiler if the parameter has not been explicitly declared. Nevertheless, if you want to declare it, you must do so in the trigger itself.
- Form objects that accept the `On Drag Over` form event - The `$0` parameter (Longint), which is the result of the `On Drag Over` form event, is typed by the compiler if the parameter has not been explicitly declared. Nevertheless, if you want to declare it, you must do so in the object method. **Note:** The compiler does not initialize the `$0` parameter. So, as soon as you use the `On Drag Over` form event, you must initialize `$0`. For example:

```
C_LONGINT($0)
If (Form event=On Drag Over)
    $0:=0
    ...
    If ($DataType=Is picture)
        $0:=-1
    End if
    ...
End if
```

## Wrong parameter type

Calling a parameter with an wrong type is an [error](#) that prevents correct execution. For example, if you write the following methods:

```
// method1
#DECLARE ($value : Text)
// method2
method1(42) //wrong type, text expected
```

This case is handled by 4D depending on the context:

- in [compiled projects](#), an error is generated at the compilation step whenever possible. Otherwise, an error is generated when the method is called.
- in interpreted projects:
  - if the parameter was declared using the [named syntax](#) (`#DECLARE` or `Function`), an error is generated when the method is called.
  - if the parameter was declared using (`C_XXX`), no error is generated, the called method receives an empty value of the expected type.

## Using object properties as named parameters

Using objects as parameters allow you to handle **named parameters**. This programming style is simple, flexible, and easy to read.

For example, using the `CreatePerson` method:

```
//CreatePerson
var $person : Object
$person:=New object("Name"; "Smith"; "Age"; 40)
ChangeAge($person)
ALERT(String($person.Age))
```

In the `ChangeAge` method you can write:

```
//ChangeAge
var $1; $para : Object
$para:=$1
$para.Age:=$para.Age+10
ALERT($para.Name+" is "+String($para.Age)+" years old.")
```

This provides a powerful way to define [optional parameters](#) (see also below). To handle missing parameters, you can either:

- check if all expected parameters are provided by comparing them to the `Null` value, or
- preset parameter values, or
- use them as empty values.

In the `ChangeAge` method above, both Age and Name properties are mandatory and would produce errors if they were missing. To avoid this case, you can just write:

```
//ChangeAge
var $1; $para : Object
$para:=$1
$para.Age:=Num($para.Age)+10
ALERT(String($para.Name)+" is "+String($para.Age)+" years old.")
```

Then both parameters are optional; if they are not filled, the result will be " is 10 years old", but no error will be generated.

Finally, with named parameters, maintaining or refactoring applications is very simple and safe. Imagine you later realize that adding 10 years is not always appropriate. You need another parameter to set how many years to add. You write:

```
$person:=New object("Name";"Smith";"Age";40;"toAdd";10)
ChangeAge($person)

//ChangeAge
var $1;$para : Object
$para:=$1
If ($para.toAdd=NULL)
    $para.toAdd:=10
End if
$para.Age:=Num($para.Age)+$para.toAdd
ALERT(String($para.Name)+" is "+String($para.Age)+" years old.")
```

The power here is that you will not need to change your existing code. It will always work as in the previous version, but if necessary, you can use another value than 10 years.

With named variables, any parameter can be optional. In the above example, all parameters are optional and anyone can be given, in any order.

## Optional parameters

In the *4D Language Reference* manual, the { } characters (braces) indicate optional parameters. For example, `ALERT (message{; okButtonTitle})` means that the `okButtonTitle` parameter may be omitted when calling the command. You can call it in the following ways:

```
ALERT("Are you sure?";"Yes I am") //2 parameters
ALERT("Time is over") //1 parameter
```

4D methods and functions also accept such optional parameters. You can declare any number of parameters. If you call a method or function with less parameters than declared, missing parameters are processed as default values in the called code, [according to their type](#). For example:

```
// "concat" function of myClass
Function concat ($param1 : Text ; $param2 : Text)->$result : Text
$result:=$param1+" "+$param2
    // Calling method
$class:=cs.myClass.new()
$class.concat("Hello") // "Hello "
$class.concat() // Displays " "
```

You can also call a method or function with more parameters than declared. They will be available within the called code through the  [`\${N}` syntax](#).

Using the `Count parameters` command from within the called method, you can detect the actual number of parameters and perform different operations depending on what you have received.

The following example displays a text message and can insert the text into a document on disk or in a 4D Write Pro area:

```
// APPEND TEXT Project Method
// APPEND TEXT ( Text { ; Text { ; Object } } )
// APPEND TEXT ( Message { ; Path { ; 4DWPArea } } )

#DECLARE ($message : Text; $path : Text; $wpArea : Object)

ALERT($message)
If(Count parameters>=3)
    WP SET TEXT($wpArea;$1;wk append)
Else
    If(Count parameters>=2)
        TEXT TO DOCUMENT($path;$message)
    End if
End if
```

After this project method has been added to your application, you can write:

```
APPEND TEXT(vtSomeText) //Will only display the message
APPEND TEXT(vtSomeText;$path) //Displays text message and appends it to
document at $path
APPEND TEXT(vtSomeText;"";$wpArea) //Displays text message and writes
it to $wpArea
```

When optional parameters are needed in your methods, you might also consider using [object properties as named parameters](#) which provide a flexible way to handle variable numbers of parameters.

## Values or references

When you pass a parameter, 4D always evaluates the parameter expression in the context of the calling method and sets the **resulting value** to the local variables in the class function or subroutine. The local variables/parameters are not the actual fields, variables, or expressions passed by the calling method; they only contain the values that have been passed. Since its scope is local, if the value of a parameter is modified in the class function/subroutine, it does not change the value in the calling method. For example:

```
//Here is some code from the method MY_METHOD
DO_SOMETHING([People]Name) //Let's say [People]Name value is "williams"
ALERT([People]Name)

//Here is the code of the method DO_SOMETHING
$1:=Uppercase($1)
ALERT($1)
```

The alert box displayed by `DO_SOMETHING` will read "WILLIAMS" and the alert box displayed by `MY_METHOD` will read "williams". The method locally changed the value of the parameter `$1`, but this does not affect the value of the field `[People]Name` passed as parameter by the method `MY_METHOD`.

There are two ways to make the method `DO_SOMETHING` change the value of the field:

1. Rather than passing the field to the method, you pass a pointer to it, so you would write:

```

//Here is some code from the method MY_METHOD
DO_SOMETHING(>[People]Name) //Let's say [People]Name value is
"williams"
ALERT([People]Last Name)

//Here the code of the method DO_SOMETHING
$1->:=Uppercase($1->)
ALERT($1->)

```

Here the parameter is not the field, but a pointer to it. Therefore, within the `DO_SOMETHING` method, `$1` is no longer the value of the field but a pointer to the field. The object **referenced** by `$1` (`$1->` in the code above) is the actual field. Consequently, changing the referenced object goes beyond the scope of the subroutine, and the actual field is affected. In this example, both alert boxes will read "WILLIAMS".

2. Rather than having the method `DO_SOMETHING` "doing something," you can rewrite the method so it returns a value. Thus you would write:

```

//Here is some code from the method MY_METHOD
[People]Name:=DO_SOMETHING([People]Name) //Let's say [People]Name
value is "williams"
ALERT([People]Name)

//Here the code of the method DO_SOMETHING
$0:=Uppercase($1)
ALERT($0)

```

This second technique of returning a value by a subroutine is called "using a function." This is described in the [Returning values](#) paragraph.

### **Particular cases: objects and collections**

You need to pay attention to the fact that Object and Collection data types can only be handled through a reference (i.e. an internal *pointer*).

Consequently, when using such data types as parameters, `$1, $2...` do not contain *values* but *references*. Modifying the value of the `$1, $2...` parameters within the subroutine will be propagated wherever the source object or collection is used. This is the same principle as for [pointers](#), except that `$1, $2...` parameters do not need to be dereferenced in the subroutine.

For example, consider the `CreatePerson` method that creates an object and sends it as a parameter:

```

//CreatePerson
var $person : Object
$person:=New object("Name";"Smith";"Age";40)
ChangeAge($person)
ALERT(String($person.Age))

```

The `ChangeAge` method adds 10 to the Age attribute of the received object

```

//ChangeAge
#DECLARE ($person : Object)
$person.Age:=$person.Age+10
ALERT(String($person.Age))

```

When you execute the `CreatePerson` method, both alert boxes will read "50" since the same object reference is handled by both methods.

**4D Server:** When parameters are passed between methods that are not executed on

the same machine (using for example the "Execute on Server" option), references are not usable. In these cases, copies of object and collection parameters are sent instead of references.

## Shared objects and collections

**Shared objects** and **shared collections** are specific [objects](#) and [collections](#) whose contents are shared between processes. In contrast to [interprocess variables](#), shared objects and shared collections have the advantage of being compatible with **preemptive 4D processes**: they can be passed by reference as parameters to commands such as `New process` or `CALL WORKER`.

Shared objects and shared collections can be stored in standard `Object` and `Collection` type variables, but must be instantiated using specific commands:

- to create a shared object, use the [New shared object](#) command,
- to create a shared collection, use the [New shared collection](#) command.

### NOTE

Shared objects and collections can be set as properties of standard (not shared) objects or collections.

In order to modify a shared object/collection, the **Use...End use** structure must be called. Reading a shared object/collection value does not require **Use...End use**.

A unique, global catalog returned by the `Storage` command is always available throughout the application and its components, and can be used to store all shared objects and collections.

## Using shared objects or collections

Once instantiated with the [New shared object](#) or [New shared collection](#) commands, shared object/collection properties and elements can be modified or read from any process of the application, under certain conditions.

### Modification

Modifications can be applied to shared objects and shared collections:

- adding or removing object properties,
- adding or editing values (provided they are supported in shared objects), including other shared objects or collections (which creates a shared group, see below).

All modification instructions in a shared object or collection require to be protected inside a [Use...End use](#) block, otherwise an error is generated.

```
$s_obj:=New shared object("prop1";"alpha")
Use($s_obj)
    $s_obj.prop1:="omega"
End Use
```

For convenience, all [collection functions](#) that modify the shared object or collection insert an internal `Use...End use` block so you do not have to code it yourself. For example:

```
$col:=New shared collection()
$col.push("alpha") //push() internally triggers Use/End use, so no
need to do it yourselves
```

If you need to execute several modifications on the same collection, you can protect all modifications with a single `Use...End use` so that modifications are performed atomically.

```
$col:=Storage.mySharedCollection
Use($col)
    $col[0]:="omega" //modifying an element requires to be performed
inside Use/End use
    $col.push("alpha") //push() internally triggers Use/End use, but
we want to do both modifications atomically
End Use
```

A shared object/collection can only be modified by one process at a time. `Use` locks the shared object/collection from other threads, while `End use` unlocks the shared object/collection (if the locking counter is at 0, see below). . Trying to modify a shared object/collection without at least one `Use...End use` generates an error. When a process calls `Use...End use` on a shared object/collection that is already in use by another process, it is simply put on hold until the `End use` unlocks it (no error is generated). Consequently, instructions within `Use...End use` structures should execute quickly and unlock the elements as soon as possible. Thus, it is strongly advised to avoid modifying a shared object or collection directly from the interface, e.g. through a dialog box.

Assigning shared objects/collections to properties or elements of other shared objects/collections is allowed and creates **shared groups**. A shared group is automatically created when a shared object/collection is set as property value or element of another shared object/collection. Shared groups allow nesting shared objects and collections but enforce additional rules:

- Calling `Use` on a shared object/collection belonging to a group locks properties/elements of all shared objects/collections of the group and increments its locking counter. Calling `End use` decrements the locking counter of the group and when the counter is at 0, all the linked shared objects/collections are unlocked.
- A shared object/collection can only belong to one shared group. An error is returned if you try to set an already grouped shared object/collection to a different group.
- Grouped shared objects/collections cannot be ungrouped. Once included in a shared group, a shared object/collection is linked permanently to that group during the whole session. Even if all references of an object/collection are removed from the parent object/collection, they will remain linked.

Please refer to example 2 for an illustration of shared group rules.

**Note:** Shared groups are managed through an internal property named *locking identifier*. For detailed information on this value, please refer to the 4D Language Reference.

## Read

Reading properties or elements of a shared object/collection is allowed without having to call the `Use...End use` structure, even if the shared object/collection is in use by another process.

However, it is necessary to read a shared object/collection within `Use...End use` when several values are linked together and must be read at once, for consistency reasons.

## Duplication

Calling `OB Copy` with a shared object (or with an object containing shared object(s) as properties) is possible, but will return a standard (not shared) object including its contained objects (if any).

## Storage

**Storage** is a unique shared object, automatically available on each application and machine. This shared object is returned by the `Storage` command. You can use this object to reference all shared objects/collections defined during the session that you want to be available from any preemptive or standard processes.

Note that, unlike standard shared objects, the `storage` object does not create a shared group when shared objects/collections are added as its properties. This exception allows the **Storage** object to be used without locking all connected shared objects or collections.

For more information, refer to the `Storage` command description.

## Use...End use

The formal syntax of the `Use...End use` structure is:

```
Use (Shared_object_or_Shared_collection)
      statement(s)
End use
```

The `Use...End use` structure defines a sequence of statements that will execute tasks on the `Shared_object_or_Shared_collection` parameter under the protection of an internal semaphore. `Shared_object_or_Shared_collection` can be any valid shared object or shared collection.

Shared objects and shared collections are designed to allow communication between processes, in particular, **preemptive 4D processes**. They can be passed by reference as parameters from a process to another one. Surrounding modifications on shared objects or shared collections by the `Use...End use` keywords is mandatory to prevent concurrent access between processes.

- Once the **Use** line is successfully executed, all `Shared_object_or_Shared_collection` properties/elements are locked for all other process in write access until the corresponding `End use` line is executed.
- The `statement(s)` sequence can execute any modification on the `Shared_object_or_Shared_collection` properties/elements without risk of concurrent access.
- If another shared object or collection is added as a property of the `Shared_object_or_Shared_collection` parameter, they become connected within the same shared group.
- If another process tries to access one of the `Shared_object_or_Shared_collection` properties or connected properties while a **Use...End use** sequence is being executed, it is automatically put on hold and waits until the current sequence is terminated.
- The **End use** line unlocks the `Shared_object_or_Shared_collection` properties and all objects of the same group.
- Several **Use...End use** structures can be nested in the 4D code. In the case of a group, each **Use** increments the locking counter of the group and each **End use** decrements it; all properties/elements will be released only when the last **End use** call sets the locking counter to 0.

## NOTE

Keep in mind that [collection functions](#) that modify shared collections automatically trigger an internal **Use** for this shared collection while the function is executed.

## Example 1

You want to launch several processes that perform an inventory task on different products and update the same shared object. The main process instantiates an empty shared object and then, launches the other processes, passing the shared object and the products to count as parameters:

```
ARRAY TEXT($_items;0)
... //fill the array with items to count
$nbItems:=Size of array($_items)
var $inventory : Object
$inventory:=New shared object
Use($inventory)
    $inventory.nbItems:=$nbItems
End use

//Create processes
For($i;1;$nbItems)
    $ps:=New
process ("HowMany";0;"HowMany_"+$_items{$i};$_items{$i};$inventory)
    // $inventory object sent by reference
End for
```

In the "HowMany" method, inventory is done and the \$inventory shared object is updated as soon as possible:

```
//HowMany
#DECLARE ($what : Text ; $inventory : Object)

$count:=CountMethod($what) //method to count products
Use($inventory) //use shared object
    $inventory[$what]:=$count //save the results for this item
End use
```

## Example 2

The following examples highlight specific rules when handling shared groups:

```
$ob1:=New shared object
$ob2:=New shared object
Use($ob1)
    $ob1.a:=$ob2 //group 1 is created
End use

$ob3:=New shared object
$ob4:=New shared object
Use($ob3)
    $ob3.a:=$ob4 //group 2 is created
End use

Use($ob1) //use an object from group 1
    $ob1.b:=$ob4 //ERROR
//$ob4 already belongs to another group
//assignment is not allowed
End use

Use($ob3)
    $ob3.a:=Null //remove any reference to $ob4 from group 2
End use

Use($ob1) //use an object from group 1
    $ob1.b:=$ob4 //ERROR
//$ob4 still belongs to group 2
//assignment is not allowed
End use
```

# Classes

## Overview

The 4D language supports the concept of **classes**. In a programming language, using a class allows you to define an object behaviour with associated properties and functions.

Once a user class is defined, you can **instantiate** objects of this class anywhere in your code. Each object is an instance of its class. A class can [extend](#) another class, and then inherits from its [functions](#) and properties ([declared](#) and [computed](#)).

The class model in 4D is similar to classes in JavaScript, and based on a chain of prototypes.

For example, you could create a `Person` class with the following definition:

```
//Class: Person.4dm
Class constructor($firstname : Text; $lastname : Text)
  This.firstName:=$firstname
  This.lastName:=$lastname

Function get fullName() -> $fullName : text
  $fullName:=This.firstName+" "+This.lastName

Function sayHello()->$welcome : Text
  $welcome:="Hello "+This.fullName
```

In a method, creating a "Person":

```
var $person : cs.Person //object of Person class
var $hello : Text
$person:=cs.Person.new("John"; "Doe")
// $person:{firstName: "John"; lastName: "Doe"; fullName: "John Doe"}
$hello:=$person.sayHello() //Hello John Doe"
```

## Managing classes

### Class definition

A user class in 4D is defined by a specific [method](#) file (.4dm), stored in the `/Project/Sources/Classes/` folder. The name of the file is the class name.

When naming classes, you should keep in mind the following rules:

- A [class name](#) must be compliant with [property naming rules](#).
- Class names are case sensitive.
- Giving the same name to a class and a database table is not recommended, in order to prevent any conflict.

For example, if you want to define a class named "Polygon", you need to create the following file:

Project folder Project Sources Classes Polygon.4dm

### Deleting a class

To delete an existing class, you can:

- on your disk, remove the .4dm class file from the "Classes" folder,
- in the 4D Explorer, select the class and click or choose **Move to Trash** from the contextual menu.

## Using the 4D interface

Class files are automatically stored at the appropriate location when created through the 4D interface, either via the **File** menu or the Explorer.

### File menu and toolbar

You can create a new class file for the project by selecting **New > Class...** in the 4D Developer **File** menu or from the toolbar.

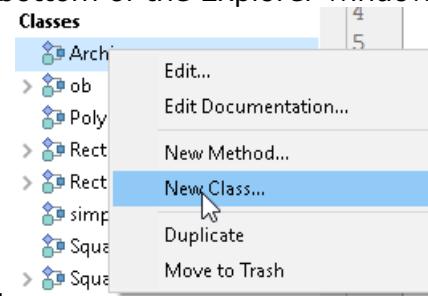
You can also use the **Ctrl+Shift+Alt+k** shortcut.

### Explorer

In the **Methods** page of the Explorer, classes are grouped in the **Classes** category.

To create a new class, you can:

- select the **Classes** category and click on the button.
- select **New Class...** from the action menu at the bottom of the Explorer window,



- or from the contextual menu of the Classes group.
- select **New > Class...** from the contextual menu of the Explorer's Home page.

### Class code support

In the various 4D windows (code editor, compiler, debugger, runtime explorer), class code is basically handled like a project method with some specificities:

- In the code editor:
  - a class cannot be run
  - a class function is a code block
  - **Goto definition** on an object member searches for class Function declarations; for example, "\$o.f()" will find "Function f".
  - **Search references** on class function declaration searches for the function used as object member; for example, "Function f" will find "\$o.f()".
- In the Runtime explorer and Debugger, class functions are displayed with the \<ClassName> constructor or \<ClassName>. \<FunctionName> format.

## Class stores

Available classes are accessible from their class stores. Two class stores are available:

- for user class store
- for built-in class store

## cs -> classStore

| Parameter Type      | Description                                   |
|---------------------|---|
| classStore object<- | User class store for the project or component |

The `cs` command returns the user class store for the current project or component. It returns all user classes [defined](#) in the opened project or component. By default, only project [ORDA classes](#) are available.

### Example

You want to create a new instance of an object of `myClass`:

```
$instance:=cs.myClass.new()
```

## 4D

## 4D -> classStore

| Parameter Type        | Description |
|-----------------------|-------------|
| classStore object<-4D | class store |

The `4D` command returns the class store for available built-in 4D classes. It provides access to specific APIs such as [CryptoKey](#).

### Example

You want to create a new key in the `CryptoKey` class:

```
$key:=4D.CryptoKey.new(New  
object("type";"ECDSA";"curve";"prime256v1"))
```

## Class object

When a class is [defined](#) in the project, it is loaded in the 4D language environment. A class is an object itself, of ["Class" class](#). A class object has the following properties and function:

- `name` string
- `superclass` object (null if none)
- `new()` function, allowing to instantiate class objects.

In addition, a class object can reference a [constructor](#) object (optional).

A class object is a [shared object](#) and can therefore be accessed from different 4D processes simultaneously.

## Inheritance

If a class inherits from another class (i.e. the [Class extends](#) keyword is used in its definition), the parent class is its [superclass](#).

When 4D does not find a function or a property in a class, it searches it in its [superclass](#); if not found, 4D continues searching in the superclass of the superclass, and so on until there is no more superclass (all objects inherit from the "Object" superclass).

# Class keywords

Specific 4D keywords can be used in class definitions:

- `Function <Name>` to define class functions of the objects.
- `Class constructor` to initialize new objects of the class.
- `property` to define static properties of the objects with a type.
- `Function get <Name>` and `Function set <Name>` to define computed properties of the objects.
- `Class extends <ClassName>` to define inheritance.

## Function

### Syntax

```
Function <name>({$parameterName : type; ...}) { ->$parameterName : type}  
// code
```

Class functions are specific properties of the class. They are objects of the [4D.Function](#) class.

In the class definition file, function declarations use the `Function` keyword, and the name of the function. The function name must be compliant with [property naming rules](#).



Starting the function name with an underscore character ("\_") will exclude the function from the autocompletion features in the 4D code editor. For example, if you declare `Function _myPrivateFunction` in `MyClass`, it will not be proposed in the code editor when you type in `"cs.MyClass. "`.

Immediately following the function name, [parameters](#) for the function can be declared with an assigned name and data type, including the return parameter (optional). For example:

```
Function computeArea($width : Integer; $height : Integer) ->$area :  
Integer
```

Within a class function, the `This` command is used as the object instance. For example:

```
Function setFullscreen($firstname : Text; $lastname : Text)  
This.firstName:=$firstname  
This.lastName:=$lastname
```

```
Function getFullscreen() ->$fullname : Text  
$fullname:=This.firstName+" "+Uppercase(This.lastName)
```

For a class function, the `Current method name` command returns: `<ClassName><FunctionName>`, for example "MyClass.myFunction".

In the application code, class functions are called as member methods of the object instance and can receive [parameters](#) if any. The following syntaxes are supported:

- use of the `()` operator. For example, `myObject.methodName("hello")`
- use of a "4D.Function" class member method:
  - [apply\(\)](#)
  - [call\(\)](#)

**Thread-safety warning:** If a class function is not thread-safe and called by a method with the "Can be run in preemptive process" attribute:

- the compiler does not generate any error (which is different compared to regular methods),
- an error is thrown by 4D only at runtime.

## Parameters

Function parameters are declared using the parameter name and the parameter type, separated by a colon. The parameter name must be compliant with [property naming rules](#). Multiple parameters (and types) are separated by semicolons (;).

```
Function add($x; $y : Variant; $z : Integer; $xy : Object)
```

If the type is not stated, the parameter will be defined as `Variant`.

The [classic 4D syntax](#) for method parameters can be used to declare class function parameters. Both syntaxes can be mixed. For example:

```
Function add($x : Integer)
var $2; $value : Integer
var $0 : Text
$value:=$x+$2
$0:=String($value)
```

## Return value

You declare the return parameter (optional) by adding an arrow (`->`) and the return parameter definition after the input parameter(s) list, or a colon (`:`) and the return parameter type only. For example:

```
Function add($x : Variant; $y : Integer)->$result : Integer
$result:=$x+$y
```

You can also declare the return parameter by adding only `: type` and use the [return expression](#) (it will also end the function execution). For example:

```
Function add($x : Variant; $y : Integer): Integer
// some code
return $x+$y
```

## Example 1

```
// Class: Rectangle
Class constructor($width : Integer; $height : Integer)
property name : Text
property height; width : Integer
This.name:="Rectangle"
This.height:=$height
This.width:=$width

// Function definition
Function getArea()->$result : Integer
$result:=(This.height)*(This.width)
```

```
// In a project method

var $rect : cs.Rectangle
var $area : Real

$rect:=cs.Rectangle.new(50;100)
$area:=$rect.getArea() //5000
```

## Example 2

This example uses the [return expression](#):

```
Function getRectArea($width : Integer; $height : Integer) : Integer
  If ($width > 0 && $height > 0)
    return $width * $height
  Else
    return 0
  End if
```

## Class Constructor

### Syntax

```
// Class: MyClass
Class Constructor({$parameterName : type; ...})
// code
```

A class constructor function accepts optional [parameters](#) and can be used to create and initialize objects of the user class.

When you call the [new\(\)](#) function, the class constructor is called with the parameters optionally passed to the [new\(\)](#) function.

There can only be one constructor function in a class (otherwise an error is returned). A constructor can use the [Super](#) keyword to call the constructor of the super class.

You can create and type instance properties inside the constructor (see example). Alternatively, if your instance properties' values do not depend on parameters passed to the constructor, you can define them using the [property](#) keyword.

## Example

```
// Class: MyClass
// Class constructor of MyClass
Class Constructor ($name : Text ; $age : Integer)
  This.name:=$name
  This.age:=$age
// In a project method
// You can instantiate an object
var $o : cs.MyClass
$o:=cs.MyClass.new("John";42)
// $o = {"name":"HelloWorld";"age":42}
```

## property

### Syntax

```
property <propertyName>{; <propertyName2>;...} { : <propertyType>}
```

The [property](#) keyword can be used to declare a property inside a user class. A class property has a name and a type.

Declaring class properties enhances code editor suggestions, type-ahead features and error detection.

Properties are declared for new objects when you call the `new()` function, however they are not automatically added to objects (they are only added when they are assigned a value).

Property names must be compliant with [property naming rules](#).

The property type can be one of the following supported types:

| propertyType               | Contents                                       |
|----------------------------|--|
| Text                       | Text value                                     |
| Date                       | Date value                                     |
| Time                       | Time value                                     |
| Boolean                    | Boolean value                                  |
| Integer                    | Long integer value                             |
| Real                       | Real value                                     |
| Pointer                    | Pointer value                                  |
| Picture                    | Picture value                                  |
| Blob                       | Scalar Blob value                              |
| Collection                 | Collection value                               |
| Variant                    | Variant value                                  |
| Object                     | Object with default class (4D.Object)          |
| 4D.<className>             | Object of the 4D class name                    |
| cs.<className>             | Object of the user class name                  |
| cs.<namespace>.<className> | Object of the <namespace> component class name |

## ⓘ INFO

The `property` keyword can only be used in class methods and outside any `Function` block.

## Example

```
// Class: MyClass  
  
property name : Text  
property age : Integer
```

In a method:

```
var $o : cs.MyClass  
$o:=cs.MyClass.new() // $o:{}  
$o.name:="John" // $o:{ "name" : "John" }  
$o.age:="Smith" // error with check syntax
```

## Function get and Function set

## Syntax

```
Function get <name> () ->$result : type  
// code  
Function set <name> ($parameterName : type)  
// code
```

`Function get` and `Function set` are accessors defining **computed properties** in the class. A computed property is a named property with a data type that masks a calculation. When a computed property value is accessed, 4D substitutes the corresponding accessor's code:

- when the property is read, the `Function get` is executed,
- when the property is written, the `Function set` is executed.

If the property is not accessed, the code never executes.

Computed properties are designed to handle data that do not necessarily need to be kept in memory. They are usually based upon persistent properties. For example, if a class object contains as persistent property the *gross price* and the *VAT rate*, the *net price* could be handled by a computed property.

In the class definition file, computed property declarations use the `Function get` (the *getter*) and `Function set` (the *setter*) keywords, followed by the name of the property. The name must be compliant with [property naming rules](#).

`Function get` returns a value of the property type and `Function set` takes a parameter of the property type. Both arguments must comply with standard [function parameters](#).

When both functions are defined, the computed property is **read-write**. If only a `Function get` is defined, the computed property is **read-only**. In this case, an error is returned if the code tries to modify the property. If only a `Function set` is defined, 4D returns *undefined* when the property is read.

The type of the computed property is defined by the `$return` type declaration of the *getter*. It can be of any [valid property type](#).

Assigning *undefined* to an object property clears its value while preserving its type. In order to do that, the `Function get` is first called to retrieve the value type, then the `Function set` is called with an empty value of that type.

## Example 1

```
//Class: Person.4dm

Class constructor($firstname : Text; $lastname : Text)
  property firstName; lastName : Text
  This.firstName:=$firstname
  This.lastName:=$lastname

  Function get fullName() -> $fullName : Text
    $fullName:=This.firstName+" "+This.lastName

  Function set fullName( $fullName : Text )
    $p:=Position(" "; $fullName)
    This.firstName:=Substring($fullName; 1; $p-1)
    This.lastName:=Substring($fullName; $p+1)
  //in a project method
  $fullName:=$person.fullName // Function get fullName() is called
  $person.fullName:="John Smith" // Function set fullName() is called
```

## Example 2

```

Function get fullAddress ()->$result : Object
    $result:=New object
    $result.fullName:=This.fullName
    $result.address:=This.address
    $result.zipCode:=This.zipCode
    $result.city:=This.city
    $result.state:=This.state
    $result.country:=This.country

```

### **Class extends <ClassName>**

## Syntax

```
// Class: ChildClass
Class extends <ParentClass>
```

The `Class extends` keyword is used in class declaration to create a user class which is a child of another user class. The child class inherits all functions of the parent class.

Class extension must respect the following rules:

- A user class cannot extend a built-in class (except 4D.Object and [ORDA classes](#) which are extended by default for user classes).
- A user class cannot extend a user class from another project or component.
- A user class cannot extend itself.
- It is not possible to extend classes in a circular way (i.e. "a" extends "b" that extends "a").

Breaking such a rule is not detected by the code editor or the interpreter, only the compiler and `check syntax` will throw an error in this case.

An extended class can call the constructor of its parent class using the [Super](#) command.

## Example

This example creates a class called `Square` from a class called `Polygon`.

```
//Class: Square
//path: Classes/Square.4dm

Class extends Polygon

Class constructor ($side : Integer)

// It calls the parent class's constructor with lengths
// provided for the Polygon's width and height
Super($side;$side)
// In derived classes, Super must be called before you
// can use 'This'
This.name:="Square"
```

```

Function getArea()
    C_LONGINT($0)
    $0:=This.height*This.width

```

[Super](#)

## Syntax

```
Super { ( param{;...;paramN} ) } { -> Object}
```

| Parameter Type | Description   |
|----------------|---|
| param          | mixed -> Parameter(s) to pass to the parent constructor |
| Result         | object <- Object's parent                               |

The `Super` keyword allows calls to the `superclass`, i.e. the parent class.

`Super` serves two different purposes:

1. Inside a [constructor code](#), `Super` is a command that allows to call the constructor of the superclass. When used in a constructor, the `Super` command appears alone and must be used before the `This` keyword is used.
  - If all class constructors in the inheritance tree are not properly called, error -10748 is generated. It's 4D developer to make sure calls are valid.
  - If the `This` command is called on an object whose superclasses have not been constructed, error -10743 is generated.
  - If `Super` is called out of an object scope, or on an object whose superclass constructor has already been called, error -10746 is generated.

```
// inside myClass constructor
var $text1; $text2 : Text
Super($text1) //calls superclass constructor with a text param
This.param:=$text2 // use second param
```

2. Inside a [class member function](#), `Super` designates the prototype of the superclass and allows to call a function of the superclass hierarchy.

```
Super.doSomething(42) //calls "doSomething" function
//declared in superclasses
```

## Example 1

This example illustrates the use of `Super` in a class constructor. The command is called to avoid duplicating the constructor parts that are common between `Rectangle` and `Square` classes.

```
// Class: Rectangle
Class constructor($width : Integer; $height : Integer)
  This.name:="Rectangle"
  This.height:=$height
  This.width:=$width

Function sayName()
  ALERT("Hi, I am a "+This.name+".")

// Function definition
Function getArea()
  var $0 : Integer
  $0:=(This.height)*(This.width)
```

```

//Class: Square

Class extends Rectangle

Class constructor ($side : Integer)

    // It calls the parent class's constructor with lengths
    // provided for the Rectangle's width and height
    Super($side;$side)
    // In derived classes, Super must be called before you
    // can use 'This'
    This.name:="Square"

Function getArea()
    C_LONGINT($0)
    $0:=This.height*This.width

```

## Example 2

This example illustrates the use of `Super` in a class member method. You created the `Rectangle` class with a function:

```

//Class: Rectangle

Function nbSides()
    var $0 : Text
    $0:="I have 4 sides"

```

You also created the `Square` class with a function calling the superclass function:

```

//Class: Square

Class extends Rectangle

Function description()
    var $0 : Text
    $0:=Super.nbSides()+" which are all equal"

```

Then you can write in a project method:

```

var $square : Object
var $message : Text
$square:=cs.Square.new()
$message:=$square.description() //I have 4 sides which are all equal

```

`This`

## Syntax

`This -> Object`

| Parameter Type | Description              |
|----------------|--------------------------|
| Result         | object <- Current object |

The `This` keyword returns a reference to the currently processed object. In 4D, it can be used in [different contexts](#).

In most cases, the value of `This` is determined by how a function is called. It can't be set by assignment during execution, and it may be different each time the function is called.

When a formula is called as a member method of an object, its `This` is set to the object the method is called on. For example:

```
$o:=New object("prop";42;"f";Formula(This.prop))  
$val:=$o.f() //42
```

When a [class constructor](#) function is used (with the [new\(\)](#) function), its `This` is bound to the new object being constructed.

```
//Class: ob  
  
Class Constructor  
  
// Create properties on This as  
// desired by assigning to them  
This.a:=42  
// in a 4D method  
$o:=cs.ob.new()  
$val:=$o.a //42
```

When calling the superclass constructor in a constructor using the [Super](#) keyword, keep in mind that `This` must not be called before the superclass constructor, otherwise an error is generated. See [this example](#).

In any cases, `This` refers to the object the method was called on, as if the method were on the object.

```
//Class: ob  
  
Function f()  
$0:=This.a+This.b
```

Then you can write in a project method:

```
$o:=cs.ob.new()  
$o.a:=5  
$o.b:=3  
$val:=$o.f() //8
```

In this example, the object assigned to the variable \$o doesn't have its own *f* property, it inherits it from its class. Since *f* is called as a method of \$o, its `This` refers to \$o.

## Class commands

Several commands of the 4D language allows you to handle class features.

### OB Class

```
OB Class ( object ) -> Object | Null
```

`OB Class` returns the class of the object passed in parameter.

### OB Instance of

```
OB Instance of ( object ; class ) -> Boolean
```

`OB Instance of` returns `true` if `object` belongs to `class` or to one of its inherited classes, and `false` otherwise.

# Commands

The 4D language contains a large number of built-in commands, allowing the developer to perform a wide range of actions.

The 4D commands are divided in two categories:

- **class functions**, which are APIs of 4D built-in classes. These functions are described in the [Class API Reference](#) section.
- **stand-alone commands**, grouped by themes. These commands are described in the *4D Language Reference* on [doc.4d.com](http://doc.4d.com). For reference, they are listed below:

[A](#) - [B](#) - [C](#) - [D](#) - [E](#) - [F](#) - [G](#) - [H](#) - [I](#) - [J](#) - [K](#) - [L](#) - [M](#) - [N](#) - [O](#) - [P](#) - [Q](#) - [R](#) - [S](#) - [T](#) - [U](#) - [V](#) - [W](#) - [X](#) - [Y](#) - [Z](#)

Name (Number)

[4D](#) (1709)

**A**

[ABORT](#) (156) - [ABORT PROCESS BY ID](#) (1634) - [Abs](#) (99) - [ACCEPT](#) (269) - [ACCUMULATE](#) (303) - [Activated](#) (346) - [Active transaction](#) (1387) - [ADD RECORD](#) (56) - [Add to date](#) (393) - [ADD TO SET](#) (119) - [ADJUST BLOBS CACHE PRIORITY](#) (1431) - [ADJUST INDEX CACHE PRIORITY](#) (1430) - [ADJUST TABLE CACHE PRIORITY](#) (1429) - [After](#) (31) - [ALERT](#) (41) - [ALL RECORDS](#) (47) - [APPEND DATA TO PASTEBOARD](#) (403) - [Append document](#) (265) - [APPEND MENU ITEM](#) (411) - [APPEND TO ARRAY](#) (911) - [APPEND TO LIST](#) (376) - [Application file](#) (491) - [Application type](#) (494) - [Application version](#) (493) - [APPLY TO SELECTION](#) (70) - [Arctan](#) (20) - [ARRAY BLOB](#) (1222) - [ARRAY BOOLEAN](#) (223) - [ARRAY DATE](#) (224) - [ARRAY INTEGER](#) (220) - [ARRAY LONGINT](#) (221) - [ARRAY OBJECT](#) (1221) - [ARRAY PICTURE](#) (279) - [ARRAY POINTER](#) (280) - [ARRAY REAL](#) (219) - [ARRAY TEXT](#) (222) - [ARRAY TIME](#) (1223) - [ARRAY TO COLLECTION](#) (1563) - [ARRAY TO LIST](#) (287) - [ARRAY TO SELECTION](#) (261) - [ASSERT](#) (1129) - [Asserted](#) (1132) - [Average](#) (2) -

**B**

[BACKUP](#) (887) - [BASE64 DECODE](#) (896) - [BASE64 ENCODE](#) (895) - [BEEP](#) (151) - [Before](#) (29) - [Before selection](#) (198) - [Begin SQL](#) (948) - [BLOB PROPERTIES](#) (536) - [BLOB size](#) (605) - [BLOB TO DOCUMENT](#) (526) - [BLOB to integer](#) (549) - [BLOB to list](#) (557) - [BLOB to longint](#) (551) - [BLOB TO PICTURE](#) (682) - [BLOB to print settings](#) (1434) - [BLOB to real](#) (553) - [BLOB to text](#) (555) - [BLOB TO USERS](#) (850) - [BLOB TO VARIABLE](#) (533) - [Bool](#) (1537) - [BOOLEAN ARRAY FROM SET](#) (646) - [BREAK LEVEL](#) (302) - [BRING TO FRONT](#) (326) - [BUILD APPLICATION](#) (871) -

**C**

[C\\_BLOB](#) (604) - [C\\_BOOLEAN](#) (305) - [C\\_COLLECTION](#) (1488) - [C\\_DATE](#) (307) - [C\\_LONGINT](#) (283) - [C\\_OBJECT](#) (1216) - [C\\_PICTURE](#) (286) - [C\\_POINTER](#) (301) - [C\\_REAL](#) (285) - [C\\_TEXT](#) (284) - [C\\_TIME](#) (306) - [C\\_VARIANT](#) (1683) - [Cache info](#) (1402) - [CALL FORM](#) (1391) - [CALL SUBFORM CONTAINER](#) (1086) - [CALL WORKER](#) (1389) - [CANCEL](#) (270) - [CANCEL TRANSACTION](#) (241) - [Caps lock down](#) (547) - [CHANGE CURRENT USER](#) (289) - [CHANGE LICENSES](#) (637) - [CHANGE PASSWORD](#) (186) - [Change string](#) (234) - [Char](#) (90) - [Character code](#) (91) - [CHECK LOG FILE](#) (799) - [Choose](#) (955) - [CLEAR LIST](#) (377) - [CLEAR NAMED SELECTION](#) (333) - [CLEAR PASTEBOARD](#) (402) - [CLEAR SEMAPHORE](#) (144) - [CLEAR SET](#) (117) - [CLEAR VARIABLE](#) (89) - [Clickcount](#) (1332) - [CLOSE DOCUMENT](#) (267) - [CLOSE PRINTING JOB](#)

(996) - [CLOSE RESOURCE FILE](#) (498) - [CLOSE WINDOW](#) (154) - [COLLECTION TO ARRAY](#) (1562) - [COMBINE PICTURES](#) (987) - [Command name](#) (538) - [Compact data file](#) (937) - [Compare strings](#) (1756) - [Compile project](#) (1760) - [COMPONENT LIST](#) (1001) - [COMPRESS BLOB](#) (534) - [CONFIRM](#) (162) - [Contextual click](#) (713) - [CONVERT COORDINATES](#) (1365) - [CONVERT FROM TEXT](#) (1011) - [Convert path POSIX to system](#) (1107) - [Convert path system to POSIX](#) (1106) - [CONVERT PICTURE](#) (1002) - [Convert to text](#) (1012) - [COPY ARRAY](#) (226) - [COPY BLOB](#) (558) - [COPY DOCUMENT](#) (541) - [Copy list](#) (626) - [COPY NAMED SELECTION](#) (331) - [Copy parameters](#) (1790) - [COPY SET](#) (600) - [Cos](#) (18) - [Count in array](#) (907) - [Count list items](#) (380) - [Count menu items](#) (405) - [Count menus](#) (404) - [Count parameters](#) (259) - [Count screens](#) (437) - [Count tasks](#) (335) - [Count user processes](#) (343) - [Count users](#) (342) - [CREATE ALIAS](#) (694) - [CREATE DATA FILE](#) (313) - [Create deployment license](#) (1811) - [Create document](#) (266) - [CREATE EMPTY SET](#) (140) - [Create entity selection](#) (1512) - [CREATE FOLDER](#) (475) - [CREATE INDEX](#) (966) - [Create menu](#) (408) - [CREATE RECORD](#) (68) - [CREATE RELATED ONE](#) (65) - [CREATE SELECTION FROM ARRAY](#) (640) - [CREATE SET](#) (116) - [CREATE SET FROM ARRAY](#) (641) - [CREATE THUMBNAIL](#) (679) - [cs](#) (1710) - [Current client authentication](#) (1355) - [Current date](#) (33) - [Current default table](#) (363) - [Current form name](#) (1298) - [Current form table](#) (627) - [Current form window](#) (827) - [Current machine](#) (483) - [Current method name](#) (684) - [Current method path](#) (1201) - [Current process](#) (322) - [Current process name](#) (1392) - [Current system user](#) (484) - [Current time](#) (178) - [Current user](#) (182) - [CUT NAMED SELECTION](#) (334) -

## D

[Data file](#) (490) - [Data file encryption status](#) (1609) - [Date](#) (102) - [Day number](#) (114) - [Day of](#) (23) - [Deactivated](#) (347) - [Dec](#) (9) - [DECRYPT BLOB](#) (690) - [Decrypt data BLOB](#) (1774) - [DEFAULT TABLE](#) (46) - [DELAY PROCESS](#) (323) - [DELETE DOCUMENT](#) (159) - [DELETE FOLDER](#) (693) - [DELETE FROM ARRAY](#) (228) - [DELETE FROM BLOB](#) (560) - [DELETE FROM LIST](#) (624) - [DELETE INDEX](#) (967) - [DELETE MENU ITEM](#) (413) - [DELETE RECORD](#) (58) - [DELETE SELECTION](#) (66) - [Delete string](#) (232) - [DELETE USER](#) (615) - [DESCRIBE QUERY EXECUTION](#) (1044) - [DIALOG](#) (40) - [DIFFERENCE](#) (122) - [DISABLE MENU ITEM](#) (150) - [Discover data key](#) (1639) - [DISPLAY NOTIFICATION](#) (910) - [DISPLAY RECORD](#) (105) - [DISPLAY SELECTION](#) (59) - [Displayed line number](#) (897) - [DISTINCT ATTRIBUTE PATHS](#) (1395) - [DISTINCT ATTRIBUTE VALUES](#) (1397) - [DISTINCT VALUES](#) (339) - [DOCUMENT LIST](#) (474) - [DOCUMENT TO BLOB](#) (525) - [Document to text](#) (1236) - [DOM Append XML child node](#) (1080) - [DOM Append XML element](#) (1082) - [DOM CLOSE XML](#) (722) - [DOM Count XML attributes](#) (727) - [DOM Count XML elements](#) (726) - [DOM Create XML element](#) (865) - [DOM Create XML element arrays](#) (1097) - [DOM Create XML Ref](#) (861) - [DOM EXPORT TO FILE](#) (862) - [DOM EXPORT TO VAR](#) (863) - [DOM Find XML element](#) (864) - [DOM Find XML element by ID](#) (1010) - [DOM Get first child XML element](#) (723) - [DOM Get last child XML element](#) (925) - [DOM Get next sibling XML element](#) (724) - [DOM Get parent XML element](#) (923) - [DOM Get previous sibling XML element](#) (924) - [DOM Get Root XML element](#) (1053) - [DOM GET XML ATTRIBUTE BY INDEX](#) (729) - [DOM GET XML ATTRIBUTE BY NAME](#) (728) - [DOM GET XML CHILD NODES](#) (1081) - [DOM Get XML document ref](#) (1088) - [DOM Get XML element](#) (725) - [DOM GET XML ELEMENT NAME](#) (730) - [DOM GET XML ELEMENT VALUE](#) (731) - [DOM Get XML information](#) (721) - [DOM Insert XML element](#) (1083) - [DOM Parse XML source](#) (719) - [DOM Parse XML variable](#) (720) - [DOM REMOVE XML ATTRIBUTE](#) (1084) - [DOM REMOVE XML ELEMENT](#) (869) - [DOM SET XML ATTRIBUTE](#) (866) - [DOM SET XML DECLARATION](#) (859) - [DOM SET XML ELEMENT NAME](#) (867) - [DOM SET XML ELEMENT VALUE](#) (868) - [DRAG WINDOW](#) (452) - [Drop position](#) (608) - [DROP REMOTE USER](#) (1633) - [ds](#) (1482) - [DUPLICATE RECORD](#) (225) - [Dynamic pop up menu](#) (1006) -

## E

[EDIT ACCESS](#) (281) - [EDIT FORMULA](#) (806) - [EDIT ITEM](#) (870) - [ENABLE MENU ITEM](#) (149) - [ENCRYPT BLOB](#) (689) - [Encrypt data BLOB](#) (1773) - [Encrypt data file](#) (1610) - [End selection](#) (36) - [End SQL](#) (949) - [Equal pictures](#) (1196) - [ERASE WINDOW](#) (160) - [Euro converter](#) (676) - [EXECUTE FORMULA](#) (63) - [EXECUTE METHOD](#) (1007) - [EXECUTE METHOD IN SUBFORM](#) (1085) - [EXECUTE ON CLIENT](#) (651) - [Execute on server](#) (373) - [Exp](#) (21) - [EXPAND BLOB](#) (535) - [EXPORT DATA](#) (666) - [EXPORT DIF](#) (84) - [EXPORT STRUCTURE](#) (1311) - [Export structure file](#) (1565) - [EXPORT SYLK](#) (85) - [EXPORT TEXT](#) (167) -

## F

[False](#) (215) - [Field](#) (253) - [Field name](#) (257) - [File](#) (1566) - [FILTER EVENT](#) (321) - [FILTER KEYSTROKE](#) (389) - [Find in array](#) (230) - [Find in field](#) (653) - [Find in list](#) (952) - [Find in sorted array](#) (1333) - [Find window](#) (449) - [FIRST RECORD](#) (50) - [FLUSH CACHE](#) (297) - [Focus object](#) (278) - [Folder](#) (1567) - [FOLDER LIST](#) (473) - [Font file](#) (1700) - [FONT LIST](#) (460) - [FONT STYLE LIST](#) (1362) - [Form](#) (1466) - [FORM Convert to dynamic](#) (1570) - [FORM EDIT](#) (1749) - [FORM Event](#) (1606) - [Form event code](#) (388) - [FORM FIRST PAGE](#) (250) - [FORM Get color scheme](#) (1761) - [FORM Get current page](#) (276) - [FORM GET ENTRY ORDER](#) (1469) - [FORM GET HORIZONTAL RESIZING](#) (1077) - [FORM GET NAMES](#) (1167) - [FORM GET OBJECTS](#) (898) - [FORM GET PROPERTIES](#) (674) - [FORM GET VERTICAL RESIZING](#) (1078) - [FORM GOTO PAGE](#) (247) - [FORM LAST PAGE](#) (251) - [FORM LOAD](#) (1103) - [FORM NEXT PAGE](#) (248) - [FORM PREVIOUS PAGE](#) (249) - [FORM SCREENSHOT](#) (940) - [FORM SET ENTRY ORDER](#) (1468) - [FORM SET HORIZONTAL RESIZING](#) (892) - [FORM SET INPUT](#) (55) - [FORM SET OUTPUT](#) (54) - [FORM SET SIZE](#) (891) - [FORM SET VERTICAL RESIZING](#) (893) - [FORM UNLOAD](#) (1299) - [Formula](#) (1597) - [Formula from string](#) (1601) - [Frontmost process](#) (327) - [Frontmost window](#) (447) -

## G

[GENERATE CERTIFICATE REQUEST](#) (691) - [Generate digest](#) (1147) - [GENERATE ENCRYPTION KEYPAIR](#) (688) - [Generate password hash](#) (1533) - [Generate UUID](#) (1066) - [Get 4D file](#) (1418) - [Get 4D folder](#) (485) - [Get action info](#) (1442) - [GET ACTIVITY SNAPSHOT](#) (1277) - [Get adjusted blobs cache priority](#) (1428) - [Get adjusted index cache priority](#) (1427) - [Get adjusted table cache priority](#) (1426) - [GET ALLOWED METHODS](#) (908) - [Get application color scheme](#) (1763) - [Get application info](#) (1599) - [Get assert enabled](#) (1130) - [GET AUTOMATIC RELATIONS](#) (899) - [GET BACKUP INFORMATION](#) (888) - [Get cache size](#) (1432) - [Get call chain](#) (1662) - [Get current printer](#) (788) - [Get database localization](#) (1009) - [Get database measures](#) (1314) - [Get database parameter](#) (643) - [Get default user](#) (826) - [GET DOCUMENT ICON](#) (700) - [Get document position](#) (481) - [GET DOCUMENT PROPERTIES](#) (477) - [Get document size](#) (479) - [Get edited text](#) (655) - [Get external data path](#) (1133) - [GET FIELD ENTRY PROPERTIES](#) (685) - [GET FIELD PROPERTIES](#) (258) - [GET FIELD RELATION](#) (920) - [GET FIELD TITLES](#) (804) - [Get file from pasteboard](#) (976) - [Get group access](#) (1738) - [GET GROUP LIST](#) (610) - [GET GROUP PROPERTIES](#) (613) - [GET HIGHLIGHT](#) (209) - [GET HIGHLIGHTED RECORDS](#) (902) - [Get indexed string](#) (510) - [GET LAST ERROR STACK](#) (1015) - [Get last field number](#) (255) - [Get last query path](#) (1045) - [Get last query plan](#) (1046) - [Get last table number](#) (254) - [Get last update log path](#) (1301) - [Get license info](#) (1489) - [Get license usage](#) (1782) - [GET LIST ITEM](#) (378) - [Get list item font](#) (954) - [GET LIST ITEM ICON](#) (951) - [GET LIST ITEM PARAMETER](#) (985) - [GET LIST ITEM PARAMETER ARRAYS](#) (1195) - [GET LIST ITEM PROPERTIES](#) (631) - [GET LIST PROPERTIES](#) (632) - [Get localized document path](#) (1105) - [Get localized string](#) (991) - [Get locked records info](#) (1316) - [GET MACRO PARAMETER](#) (997) - [GET MEMORY STATISTICS](#) (1118) - [Get menu bar reference](#) (979) - [Get menu item](#) (422) - [GET MENU ITEM ICON](#) (983) - [Get menu item key](#) (424) - [Get menu item mark](#) (428) - [Get menu item method](#) (981) - [Get menu item modifiers](#) (980) - [Get menu item parameter](#) (1003) - [GET MENU ITEM PROPERTY](#) (972) - [Get menu item style](#) (426) -

[GET MENU ITEMS](#) (977) - [Get menu title](#) (430) - [GET MISSING TABLE NAMES](#) (1125)  
- [Get Monitored Activity](#) (1713) - [GET MOUSE](#) (468) - [GET PASTEBOARD DATA](#) (401) -  
[GET PASTEBOARD DATA TYPE](#) (958) - [Get picture file name](#) (1171) - [GET PICTURE  
FORMATS](#) (1406) - [GET PICTURE FROM LIBRARY](#) (565) - [GET PICTURE FROM  
PASTEBOARD](#) (522) - [GET PICTURE KEYWORDS](#) (1142) - [GET PICTURE METADATA](#)  
(1122) - [GET PICTURE RESOURCE](#) (502) - [Get plugin access](#) (846) - [Get pointer](#) (304)  
- [Get print marker](#) (708) - [GET PRINT OPTION](#) (734) - [Get print preview](#) (1197) - [GET  
PRINTABLE AREA](#) (703) - [GET PRINTABLE MARGIN](#) (711) - [Get printed height](#) (702) -  
[Get process activity](#) (1495) - [GET PROCESS VARIABLE](#) (371) - [GET QUERY  
DESTINATION](#) (1155) - [Get query limit](#) (1156) - [GET REGISTERED CLIENTS](#) (650) -  
[GET RELATION PROPERTIES](#) (686) - [GET RESOURCE](#) (508) - [Get resource name](#) (513)  
- [Get resource properties](#) (515) - [GET RESTORE INFORMATION](#) (889) - [Get selected  
menu item parameter](#) (1005) - [GET SERIAL INFORMATION](#) (696) - [GET SERIAL PORT  
MAPPING](#) (909) - [Get string resource](#) (506) - [GET STYLE SHEET INFO](#) (1256) - [Get  
subrecord key](#) (1137) - [GET SYSTEM FORMAT](#) (994) - [Get system info](#) (1571) - [Get  
table fragmentation](#) (1127) - [GET TABLE PROPERTIES](#) (687) - [GET TABLE TITLES](#)  
(803) - [Get text from pasteboard](#) (524) - [GET TEXT KEYWORDS](#) (1141) - [Get text  
resource](#) (504) - [GET USER LIST](#) (609) - [GET USER PROPERTIES](#) (611) - [GET  
WINDOW RECT](#) (443) - [Get window title](#) (450) - [GOTO OBJECT](#) (206) - [GOTO RECORD](#)  
(242) - [GOTO SELECTED RECORD](#) (245) - [GOTO XY](#) (161) - [GRAPH](#) (169) - [GRAPH  
SETTINGS](#) (298) -

## H

[HIDE MENU BAR](#) (432) - [HIDE PROCESS](#) (324) - [HIDE TOOL BAR](#) (434) - [HIDE  
WINDOW](#) (436) - [HIGHLIGHT RECORDS](#) (656) - [HIGHLIGHT TEXT](#) (210) - [HTTP  
AUTHENTICATE](#) (1161) - [HTTP Get](#) (1157) - [HTTP Get certificates folder](#) (1307) -  
[HTTP GET OPTION](#) (1159) - [HTTP Request](#) (1158) - [HTTP SET CERTIFICATES FOLDER](#)  
(1306) - [HTTP SET OPTION](#) (1160) -

## I

[IDLE](#) (311) - [IMAP New transporter](#) (1723) - [IMPORT DATA](#) (665) - [IMPORT DIF](#) (86) -  
[IMPORT STRUCTURE](#) (1310) - [IMPORT SYLK](#) (87) - [IMPORT TEXT](#) (168) - [In break](#)  
(113) - [In footer](#) (191) - [In header](#) (112) - [In transaction](#) (397) - [INSERT IN ARRAY](#)  
(227) - [INSERT IN BLOB](#) (559) - [INSERT IN LIST](#) (625) - [INSERT MENU ITEM](#) (412) -  
[Insert string](#) (231) - [Int](#) (8) - [INTEGER TO BLOB](#) (548) - [INTEGRATE MIRROR LOG  
FILE](#) (1312) - [INTERSECTION](#) (121) - [INVOKER ACTION](#) (1439) - [Is a list](#) (621) - [Is a  
variable](#) (294) - [Is compiled mode](#) (492) - [Is data file locked](#) (716) - [Is editing text](#)  
(1744) - [Is field number valid](#) (1000) - [Is field value Null](#) (964) - [Is in print preview](#)  
(1198) - [Is in set](#) (273) - [Is license available](#) (714) - [Is macOS](#) (1572) - [Is new record](#)  
(668) - [Is nil pointer](#) (315) - [Is picture file](#) (1113) - [Is record loaded](#) (669) - [Is table  
number valid](#) (999) - [Is user deleted](#) (616) - [Is waiting mouse up](#) (1422) - [Is Windows](#)  
(1573) -

## J

[JSON Parse](#) (1218) - [JSON PARSE ARRAY](#) (1219) - [JSON Resolve pointers](#) (1478) -  
[JSON Stringify](#) (1217) - [JSON Stringify array](#) (1228) - [JSON TO SELECTION](#) (1235) -  
[JSON Validate](#) (1456) -

## K

[Keystroke](#) (390) - [KILL WORKER](#) (1390) -

## L

[Last errors](#) (1799) - [LAST RECORD](#) (200) - [LAUNCH EXTERNAL PROCESS](#) (811) -  
[LDAP LOGIN](#) (1326) - [LDAP LOGOUT](#) (1327) - [LDAP Search](#) (1328) - [LDAP SEARCH](#)

[ALL](#) (1329) - [Length](#) (16) - [Level](#) (101) - [List item parent](#) (633) - [List item position](#) (629) - [LIST OF CHOICE LISTS](#) (957) - [LIST OF STYLE SHEETS](#) (1255) - [LIST TO ARRAY](#) (288) - [LIST TO BLOB](#) (556) - [LISTBOX COLLAPSE](#) (1101) - [LISTBOX DELETE COLUMN](#) (830) - [LISTBOX DELETE ROWS](#) (914) - [LISTBOX DUPLICATE COLUMN](#) (1273) - [LISTBOX EXPAND](#) (1100) - [LISTBOX Get array](#) (1278) - [LISTBOX GET ARRAYS](#) (832) - [LISTBOX Get auto row height](#) (1502) - [LISTBOX GET CELL COORDINATES](#) (1330) - [LISTBOX GET CELL POSITION](#) (971) - [LISTBOX Get column formula](#) (1202) - [LISTBOX Get column width](#) (834) - [LISTBOX Get footer calculation](#) (1150) - [LISTBOX Get footers height](#) (1146) - [LISTBOX GET GRID](#) (1199) - [LISTBOX GET GRID COLORS](#) (1200) - [LISTBOX Get headers height](#) (1144) - [LISTBOX GET HIERARCHY](#) (1099) - [LISTBOX Get locked columns](#) (1152) - [LISTBOX Get number of columns](#) (831) - [LISTBOX Get number of rows](#) (915) - [LISTBOX GET OBJECTS](#) (1302) - [LISTBOX GET PRINT INFORMATION](#) (1110) - [LISTBOX Get property](#) (917) - [LISTBOX Get row color](#) (1658) - [LISTBOX Get row color as number](#) (1271) - [LISTBOX Get row font style](#) (1269) - [LISTBOX Get row height](#) (1408) - [LISTBOX Get rows height](#) (836) - [LISTBOX Get static columns](#) (1154) - [LISTBOX GET TABLE SOURCE](#) (1014) - [LISTBOX INSERT COLUMN](#) (829) - [LISTBOX INSERT COLUMN FORMULA](#) (970) - [LISTBOX INSERT ROWS](#) (913) - [LISTBOX MOVE COLUMN](#) (1274) - [LISTBOX MOVED COLUMN NUMBER](#) (844) - [LISTBOX MOVED ROW NUMBER](#) (837) - [LISTBOX SELECT BREAK](#) (1117) - [LISTBOX SELECT ROW](#) (912) - [LISTBOX SELECT ROWS](#) (1715) - [LISTBOX SET ARRAY](#) (1279) - [LISTBOX SET AUTO ROW HEIGHT](#) (1501) - [LISTBOX SET COLUMN FORMULA](#) (1203) - [LISTBOX SET COLUMN WIDTH](#) (833) - [LISTBOX SET FOOTER CALCULATION](#) (1140) - [LISTBOX SET FOOTERS HEIGHT](#) (1145) - [LISTBOX SET GRID](#) (841) - [LISTBOX SET GRID COLOR](#) (842) - [LISTBOX SET HEADERS HEIGHT](#) (1143) - [LISTBOX SET HIERARCHY](#) (1098) - [LISTBOX SET LOCKED COLUMNS](#) (1151) - [LISTBOX SET PROPERTY](#) (1440) - [LISTBOX SET ROW COLOR](#) (1270) - [LISTBOX SET ROW FONT STYLE](#) (1268) - [LISTBOX SET ROW HEIGHT](#) (1409) - [LISTBOX SET ROWS HEIGHT](#) (835) - [LISTBOX SET STATIC COLUMNS](#) (1153) - [LISTBOX SET TABLE SOURCE](#) (1013) - [LISTBOX SORT COLUMNS](#) (916) - [Load 4D View document](#) (1528) - [Load list](#) (383) - [LOAD RECORD](#) (52) - [LOAD SET](#) (185) - [LOAD VARIABLES](#) (74) - [Locked](#) (147) - [LOCKED BY](#) (353) - [Log](#) (22) - [LOG EVENT](#) (667) - [Log File](#) (928) - [LOG FILE TO JSON](#) (1352) - [LONGINT ARRAY FROM SELECTION](#) (647) - [LONGINT TO BLOB](#) (550) - [Lowercase](#) (14) -

## M

[Macintosh command down](#) (546) - [Macintosh control down](#) (544) - [Macintosh option down](#) (545) - [MAIL Convert from MIME](#) (1681) - [MAIL Convert to MIME](#) (1604) - [MAIL New attachment](#) (1644) - [Match regex](#) (1019) - [Max](#) (3) - [MAXIMIZE WINDOW](#) (453) - [Menu bar height](#) (440) - [Menu bar screen](#) (441) - [Menu selected](#) (152) - [MESSAGE](#) (88) - [MESSAGES OFF](#) (175) - [MESSAGES ON](#) (181) - [Method called on error](#) (704) - [Method called on event](#) (705) - [METHOD Get attribute](#) (1169) - [METHOD GET ATTRIBUTES](#) (1334) - [METHOD GET CODE](#) (1190) - [METHOD GET COMMENTS](#) (1189) - [METHOD GET FOLDERS](#) (1206) - [METHOD GET MODIFICATION DATE](#) (1170) - [METHOD GET NAMES](#) (1166) - [METHOD Get path](#) (1164) - [METHOD GET PATHS](#) (1163) - [METHOD GET PATHS FORM](#) (1168) - [METHOD OPEN PATH](#) (1213) - [METHOD RESOLVE PATH](#) (1165) - [METHOD SET ACCESS MODE](#) (1191) - [METHOD SET ATTRIBUTE](#) (1192) - [METHOD SET ATTRIBUTES](#) (1335) - [METHOD SET CODE](#) (1194) - [METHOD SET COMMENTS](#) (1193) - [Milliseconds](#) (459) - [Min](#) (4) - [MINIMIZE WINDOW](#) (454) - [MOBILE APP REFRESH SESSIONS](#) (1596) - [Mod](#) (98) - [Modified](#) (32) - [Modified record](#) (314) - [MODIFY RECORD](#) (57) - [MODIFY SELECTION](#) (204) - [Month of](#) (24) - [MOVE DOCUMENT](#) (540) - [MULTI SORT ARRAY](#) (718) -

## N

[New collection](#) (1472) - [New data key](#) (1611) - [New list](#) (375) - [New log file](#) (926) - [New object](#) (1471) - [New process](#) (317) - [New shared collection](#) (1527) - [New shared](#)

[object](#) (1526) - [New signal](#) (1641) - [NEXT RECORD](#) (51) - [Next window](#) (448) - [NO DEFAULT TABLE](#) (993) - [Not](#) (34) - [NOTIFY RESOURCES FOLDER MODIFICATION](#) (1052) - [Null](#) (1517) - [Num](#) (11) -

## O

[OB Class](#) (1730) - [OB Copy](#) (1225) - [OB Entries](#) (1720) - [OB Get](#) (1224) - [OB GET ARRAY](#) (1229) - [OB GET PROPERTY NAMES](#) (1232) - [OB Get type](#) (1230) - [OB Instance of](#) (1731) - [OB Is defined](#) (1231) - [OB Is empty](#) (1297) - [OB Is shared](#) (1759) - [OB Keys](#) (1719) - [OB REMOVE](#) (1226) - [OB SET](#) (1220) - [OB SET ARRAY](#) (1227) - [OB SET NULL](#) (1233) - [OB Values](#) (1718) - [OBJECT DUPLICATE](#) (1111) - [OBJECT Get action](#) (1457) - [OBJECT Get auto spellcheck](#) (1174) - [OBJECT GET BEST SIZE](#) (717) - [OBJECT Get border style](#) (1263) - [OBJECT Get context menu](#) (1252) - [OBJECT GET COORDINATES](#) (663) - [OBJECT Get corner radius](#) (1324) - [OBJECT Get data source](#) (1265) - [OBJECT GET DRAG AND DROP OPTIONS](#) (1184) - [OBJECT Get enabled](#) (1079) - [OBJECT Get enterable](#) (1067) - [OBJECT GET EVENTS](#) (1238) - [OBJECT Get filter](#) (1073) - [OBJECT Get focus rectangle invisible](#) (1178) - [OBJECT Get font](#) (1069) - [OBJECT Get font size](#) (1070) - [OBJECT Get font style](#) (1071) - [OBJECT Get format](#) (894) - [OBJECT Get help tip](#) (1182) - [OBJECT Get horizontal alignment](#) (707) - [OBJECT Get indicator type](#) (1247) - [OBJECT Get keyboard layout](#) (1180) - [OBJECT Get list name](#) (1072) - [OBJECT Get list reference](#) (1267) - [OBJECT GET MAXIMUM VALUE](#) (1245) - [OBJECT GET MINIMUM VALUE](#) (1243) - [OBJECT Get multiline](#) (1254) - [OBJECT Get name](#) (1087) - [OBJECT Get placeholder](#) (1296) - [OBJECT Get pointer](#) (1124) - [OBJECT GET PRINT VARIABLE FRAME](#) (1241) - [OBJECT GET RESIZING OPTIONS](#) (1176) - [OBJECT GET RGB COLORS](#) (1074) - [OBJECT GET SCROLL POSITION](#) (1114) - [OBJECT GET SCROLLBAR](#) (1076) - [OBJECT GET SHORTCUT](#) (1186) - [OBJECT Get style sheet](#) (1258) - [OBJECT GET SUBFORM](#) (1139) - [OBJECT GET SUBFORM CONTAINER SIZE](#) (1148) - [OBJECT Get subform container value](#) (1785) - [OBJECT Get text orientation](#) (1283) - [OBJECT Get three states checkbox](#) (1250) - [OBJECT Get title](#) (1068) - [OBJECT Get type](#) (1300) - [OBJECT Get value](#) (1743) - [OBJECT Get vertical alignment](#) (1188) - [OBJECT Get visible](#) (1075) - [OBJECT Is styled text](#) (1261) - [OBJECT MOVE](#) (664) - [OBJECT SET ACTION](#) (1259) - [OBJECT SET AUTO SPELLCHECK](#) (1173) - [OBJECT SET BORDER STYLE](#) (1262) - [OBJECT SET CONTEXT MENU](#) (1251) - [OBJECT SET COORDINATES](#) (1248) - [OBJECT SET CORNER RADIUS](#) (1323) - [OBJECT SET DATA SOURCE](#) (1264) - [OBJECT SET DRAG AND DROP OPTIONS](#) (1183) - [OBJECT SET ENABLED](#) (1123) - [OBJECT SET ENTERABLE](#) (238) - [OBJECT SET EVENTS](#) (1239) - [OBJECT SET FILTER](#) (235) - [OBJECT SET FOCUS RECTANGLE INVISIBLE](#) (1177) - [OBJECT SET FONT](#) (164) - [OBJECT SET FONT SIZE](#) (165) - [OBJECT SET FONT STYLE](#) (166) - [OBJECT SET FORMAT](#) (236) - [OBJECT SET HELP TIP](#) (1181) - [OBJECT SET HORIZONTAL ALIGNMENT](#) (706) - [OBJECT SET INDICATOR TYPE](#) (1246) - [OBJECT SET KEYBOARD LAYOUT](#) (1179) - [OBJECT SET LIST BY NAME](#) (237) - [OBJECT SET LIST BY REFERENCE](#) (1266) - [OBJECT SET MAXIMUM VALUE](#) (1244) - [OBJECT SET MINIMUM VALUE](#) (1242) - [OBJECT SET MULTILINE](#) (1253) - [OBJECT SET PLACEHOLDER](#) (1295) - [OBJECT SET PRINT VARIABLE FRAME](#) (1240) - [OBJECT SET RESIZING OPTIONS](#) (1175) - [OBJECT SET RGB COLORS](#) (628) - [OBJECT SET SCROLL POSITION](#) (906) - [OBJECT SET SCROLLBAR](#) (843) - [OBJECT SET SHORTCUT](#) (1185) - [OBJECT SET STYLE SHEET](#) (1257) - [OBJECT SET SUBFORM](#) (1138) - [OBJECT SET SUBFORM CONTAINER VALUE](#) (1784) - [OBJECT SET TEXT ORIENTATION](#) (1284) - [OBJECT SET THREE STATES CHECKBOX](#) (1249) - [OBJECT SET TITLE](#) (194) - [OBJECT SET VALUE](#) (1742) - [OBJECT SET VERTICAL ALIGNMENT](#) (1187) - [OBJECT SET VISIBLE](#) (603) - [Object to path](#) (1548) - [Old](#) (35) - [OLD RELATED MANY](#) (263) - [OLD RELATED ONE](#) (44) - [ON ERR CALL](#) (155) - [ON EVENT CALL](#) (190) - [On REST Authentication database method](#) (3367) - [ONE RECORD SELECT](#) (189) - [OPEN ADMINISTRATION WINDOW](#) (1047) - [OPEN COLOR PICKER](#) (1304) - [OPEN DATA FILE](#) (312) - [OPEN DATABASE](#) (1321) - [Open datastore](#) (1452) - [Open document](#) (264) - [OPEN FONT PICKER](#) (1303) - [Open form window](#) (675) - [OPEN PRINTING JOB](#) (995) - [Open](#)

[resource file](#) (497) - [OPEN RUNTIME EXPLORER](#) (1781) - [OPEN SECURITY CENTER](#) (1018) - [OPEN SETTINGS WINDOW](#) (903) - [OPEN URL](#) (673) - [Open window](#) (153) - [ORDER BY](#) (49) - [ORDER BY ATTRIBUTE](#) (1407) - [ORDER BY FORMULA](#) (300) - [Outside call](#) (328) -

## P

[PAGE BREAK](#) (6) - [Parse formula](#) (1576) - [Pasteboard data size](#) (400) - [Path to object](#) (1547) - [PAUSE INDEXES](#) (1293) - [PAUSE PROCESS](#) (319) - [PHP Execute](#) (1058) - [PHP GET FULL RESPONSE](#) (1061) - [PHP GET OPTION](#) (1060) - [PHP SET OPTION](#) (1059) - [PICTURE CODEC LIST](#) (992) - [PICTURE LIBRARY LIST](#) (564) - [PICTURE PROPERTIES](#) (457) - [Picture size](#) (356) - [PICTURE TO BLOB](#) (692) - [PLAY](#) (290) - [PLUGIN LIST](#) (847) - [POP RECORD](#) (177) - [Pop up menu](#) (542) - [POP3 New transporter](#) (1697) - [Position](#) (15) - [POST CLICK](#) (466) - [POST EVENT](#) (467) - [POST KEY](#) (465) - [POST OUTSIDE CALL](#) (329) - [PREVIOUS RECORD](#) (110) - [Print form](#) (5) - [PRINT LABEL](#) (39) - [Print object](#) (1095) - [PRINT OPTION VALUES](#) (785) - [PRINT RECORD](#) (71) - [PRINT SELECTION](#) (60) - [PRINT SETTINGS](#) (106) - [Print settings to BLOB](#) (1433) - [PRINTERS LIST](#) (789) - [Printing page](#) (275) - [PROCESS 4D TAGS](#) (816) - [Process aborted](#) (672) - [Process number](#) (372) - [PROCESS PROPERTIES](#) (336) - [Process state](#) (330) - [Progress New](#) (0) - [PUSH RECORD](#) (176) -

## Q

[QR BLOB TO REPORT](#) (771) - [QR Count columns](#) (764) - [QR DELETE COLUMN](#) (749) - [QR DELETE OFFSCREEN AREA](#) (754) - [QR EXECUTE COMMAND](#) (791) - [QR Find column](#) (776) - [QR Get area property](#) (795) - [QR GET BORDERS](#) (798) - [QR Get command status](#) (792) - [QR GET DESTINATION](#) (756) - [QR Get document property](#) (773) - [QR Get drop column](#) (747) - [QR GET HEADER AND FOOTER](#) (775) - [QR Get HTML template](#) (751) - [QR GET INFO COLUMN](#) (766) - [QR Get info row](#) (769) - [QR Get report kind](#) (755) - [QR Get report table](#) (758) - [QR GET SELECTION](#) (793) - [QR GET SORTS](#) (753) - [QR Get text property](#) (760) - [QR GET TOTALS DATA](#) (768) - [QR GET TOTALS SPACING](#) (762) - [QR INSERT COLUMN](#) (748) - [QR MOVE COLUMN](#) (1325) - [QR NEW AREA](#) (1320) - [QR New offscreen area](#) (735) - [QR ON COMMAND](#) (790) - [QR REPORT](#) (197) - [QR REPORT TO BLOB](#) (770) - [QR RUN](#) (746) - [QR SET AREA PROPERTY](#) (796) - [QR SET BORDERS](#) (797) - [QR SET DESTINATION](#) (745) - [QR SET DOCUMENT PROPERTY](#) (772) - [QR SET HEADER AND FOOTER](#) (774) - [QR SET HTML TEMPLATE](#) (750) - [QR SET INFO COLUMN](#) (765) - [QR SET INFO ROW](#) (763) - [QR SET REPORT KIND](#) (738) - [QR SET REPORT TABLE](#) (757) - [QR SET SELECTION](#) (794) - [QR SET SORTS](#) (752) - [QR SET TEXT PROPERTY](#) (759) - [QR SET TOTALS DATA](#) (767) - [QR SET TOTALS SPACING](#) (761) - [QUERY](#) (277) - [QUERY BY ATTRIBUTE](#) (1331) - [QUERY BY EXAMPLE](#) (292) - [QUERY BY FORMULA](#) (48) - [QUERY BY SQL](#) (942) - [QUERY SELECTION](#) (341) - [QUERY SELECTION BY ATTRIBUTE](#) (1424) - [QUERY SELECTION BY FORMULA](#) (207) - [QUERY SELECTION WITH ARRAY](#) (1050) - [QUERY WITH ARRAY](#) (644) - [QUIT 4D](#) (291) -

## R

[Random](#) (100) - [READ ONLY](#) (145) - [Read only state](#) (362) - [READ PICTURE FILE](#) (678) - [READ WRITE](#) (146) - [REAL TO BLOB](#) (552) - [RECEIVE BUFFER](#) (172) - [RECEIVE PACKET](#) (104) - [RECEIVE RECORD](#) (79) - [RECEIVE VARIABLE](#) (81) - [Record number](#) (243) - [Records in selection](#) (76) - [Records in set](#) (195) - [Records in table](#) (83) - [REDRAW](#) (174) - [REDRAW WINDOW](#) (456) - [REDUCE SELECTION](#) (351) - [Refresh license](#) (1336) - [REGENERATE MISSING TABLE](#) (1126) - [REGISTER CLIENT](#) (648) - [Register data key](#) (1638) - [REJECT](#) (38) - [REJECT NEW REMOTE CONNECTIONS](#) (1635) - [RELATE MANY](#) (262) - [RELATE MANY SELECTION](#) (340) - [RELATE ONE](#) (42) - [RELATE ONE SELECTION](#) (349) - [RELEASE MENU](#) (978) - [RELOAD EXTERNAL DATA](#) (1135) - [RELOAD PROJECT](#) (1739) - [REMOVE FROM SET](#) (561) - [REMOVE PICTURE FROM LIBRARY](#) (567) - [Replace string](#) (233) - [Request](#) (163) -

[RESIZE FORM WINDOW](#) (890) - [RESOLVE ALIAS](#) (695) - [RESOLVE POINTER](#) (394) - [RESOURCE LIST](#) (500) - [RESOURCE TYPE LIST](#) (499) - [RESTART 4D](#) (1292) - [RESTORE](#) (918) - [RESUME INDEXES](#) (1294) - [RESUME PROCESS](#) (320) - [RESUME TRANSACTION](#) (1386) - [Right click](#) (712) - [Round](#) (94) -

S

[SAVE LIST](#) (384) - [SAVE RECORD](#) (53) - [SAVE RELATED ONE](#) (43) - [SAVE SET](#) (184) - [SAVE VARIABLES](#) (75) - [SAX ADD PROCESSING INSTRUCTION](#) (857) - [SAX ADD XML CDATA](#) (856) - [SAX ADD XML COMMENT](#) (852) - [SAX ADD XML DOCTYPE](#) (851) - [SAX ADD XML ELEMENT VALUE](#) (855) - [SAX CLOSE XML ELEMENT](#) (854) - [SAX GET XML CDATA](#) (878) - [SAX GET XML COMMENT](#) (874) - [SAX GET XML DOCUMENT VALUES](#) (873) - [SAX GET XML ELEMENT](#) (876) - [SAX GET XML ELEMENT VALUE](#) (877) - [SAX GET XML ENTITY](#) (879) - [SAX Get XML node](#) (860) - [SAX GET XML PROCESSING INSTRUCTION](#) (875) - [SAX OPEN XML ELEMENT](#) (853) - [SAX OPEN XML ELEMENT ARRAYS](#) (921) - [SAX SET XML DECLARATION](#) (858) - [SCAN INDEX](#) (350) - [SCREEN COORDINATES](#) (438) - [SCREEN DEPTH](#) (439) - [Screen height](#) (188) - [Screen width](#) (187) - [Select document](#) (905) - [Select folder](#) (670) - [SELECT LIST ITEMS BY POSITION](#) (381) - [SELECT LIST ITEMS BY REFERENCE](#) (630) - [SELECT LOG FILE](#) (345) - [Select RGB Color](#) (956) - [Selected list items](#) (379) - [Selected record number](#) (246) - [SELECTION RANGE TO ARRAY](#) (368) - [SELECTION TO ARRAY](#) (260) - [Selection to JSON](#) (1234) - [Self](#) (308) - [Semaphore](#) (143) - [SEND MESSAGE TO REMOTE USER](#) (1632) - [SEND PACKET](#) (103) - [SEND RECORD](#) (78) - [SEND VARIABLE](#) (80) - [Sequence number](#) (244) - [Session](#) (1714) - [SET ABOUT](#) (316) - [SET ALLOWED METHODS](#) (805) - [SET APPLICATION COLOR SCHEME](#) (1762) - [SET ASSERT ENABLED](#) (1131) - [SET AUTOMATIC RELATIONS](#) (310) - [SET BLOB SIZE](#) (606) - [SET BLOBS CACHE PRIORITY](#) (1425) - [SET CACHE SIZE](#) (1399) - [SET CHANNEL](#) (77) - [SET CURRENT PRINTER](#) (787) - [SET CURSOR](#) (469) - [SET DATABASE LOCALIZATION](#) (1104) - [SET DATABASE PARAMETER](#) (642) - [SET DEFAULT CENTURY](#) (392) - [SET DOCUMENT POSITION](#) (482) - [SET DOCUMENT PROPERTIES](#) (478) - [SET DOCUMENT SIZE](#) (480) - [SET DRAG ICON](#) (1272) - [SET ENVIRONMENT VARIABLE](#) (812) - [SET EXTERNAL DATA PATH](#) (1134) - [SET FIELD RELATION](#) (919) - [SET FIELD TITLES](#) (602) - [SET FIELD VALUE NULL](#) (965) - [SET FILE TO PASTEBOARD](#) (975) - [SET GROUP ACCESS](#) (1737) - [Set group properties](#) (614) - [SET HELP MENU](#) (1801) - [SET INDEX](#) (344) - [SET INDEX CACHE PRIORITY](#) (1401) - [SET LIST ITEM](#) (385) - [SET LIST ITEM FONT](#) (953) - [SET LIST ITEM ICON](#) (950) - [SET LIST ITEM PARAMETER](#) (986) - [SET LIST ITEM PROPERTIES](#) (386) - [SET LIST PROPERTIES](#) (387) - [SET MACRO PARAMETER](#) (998) - [SET MENU BAR](#) (67) - [SET MENU ITEM](#) (348) - [SET MENU ITEM ICON](#) (984) - [SET MENU ITEM MARK](#) (208) - [SET MENU ITEM METHOD](#) (982) - [SET MENU ITEM PARAMETER](#) (1004) - [SET MENU ITEM PROPERTY](#) (973) - [SET MENU ITEM SHORTCUT](#) (423) - [SET MENU ITEM STYLE](#) (425) - [SET PICTURE FILE NAME](#) (1172) - [SET PICTURE METADATA](#) (1121) - [SET PICTURE TO LIBRARY](#) (566) - [SET PICTURE TO PASTEBOARD](#) (521) - [SET PLUGIN ACCESS](#) (845) - [SET PRINT MARKER](#) (709) - [SET PRINT OPTION](#) (733) - [SET PRINT PREVIEW](#) (364) - [SET PRINTABLE MARGIN](#) (710) - [SET PROCESS VARIABLE](#) (370) - [SET QUERY AND LOCK](#) (661) - [SET QUERY DESTINATION](#) (396) - [SET QUERY LIMIT](#) (395) - [SET REAL COMPARISON LEVEL](#) (623) - [SET RECENT FONTS](#) (1305) - [SET SCREEN DEPTH](#) (537) - [SET TABLE CACHE PRIORITY](#) (1400) - [SET TABLE TITLES](#) (601) - [SET TEXT TO PASTEBOARD](#) (523) - [SET TIMEOUT](#) (268) - [SET TIMER](#) (645) - [SET UPDATE FOLDER](#) (1291) - [SET USER ALIAS](#) (1666) - [Set user properties](#) (612) - [SET WINDOW RECT](#) (444) - [SET WINDOW TITLE](#) (213) - [Shift down](#) (543) - [SHOW MENU BAR](#) (431) - [SHOW ON DISK](#) (922) - [SHOW PROCESS](#) (325) - [SHOW TOOL BAR](#) (433) - [SHOW WINDOW](#) (435) - [Sin](#) (17) - [Size of array](#) (274) - [SMTP New transporter](#) (1608) - [SOAP DECLARATION](#) (782) - [SOAP Get info](#) (784) - [SOAP REJECT NEW REQUESTS](#) (1636) - [SOAP Request](#) (783) - [SOAP SEND FAULT](#) (781) - [SORT ARRAY](#) (229) - [SORT LIST](#) (391) - [SPELL ADD TO USER DICTIONARY](#) (1214) - [SPELL CHECK TEXT](#) (1215) - [SPELL CHECKING](#) (900) - [SPELL Get current dictionary](#) (1205) - [SPELL GET DICTIONARY LIST](#) (1204) -

[SPELL SET CURRENT DICTIONARY](#) (904) - [Split string](#) (1554) - [SQL CANCEL LOAD](#) (824) - [SQL End selection](#) (821) - [SQL EXECUTE](#) (820) - [SQL EXECUTE SCRIPT](#) (1089) - [SQL EXPORT DATABASE](#) (1065) - [SQL EXPORT SELECTION](#) (1064) - [SQL Get current data source](#) (990) - [SQL GET DATA SOURCE LIST](#) (989) - [SQL GET LAST ERROR](#) (825) - [SQL GET OPTION](#) (819) - [SQL LOAD RECORD](#) (822) - [SQL LOGIN](#) (817) - [SQL LOGOUT](#) (872) - [SQL SET OPTION](#) (818) - [SQL SET PARAMETER](#) (823) - [Square root](#) (539) - [ST COMPUTE EXPRESSIONS](#) (1285) - [ST FREEZE EXPRESSIONS](#) (1282) - [ST GET ATTRIBUTES](#) (1094) - [ST Get content type](#) (1286) - [ST Get expression](#) (1287) - [ST GET OPTIONS](#) (1290) - [ST Get plain text](#) (1092) - [ST Get text](#) (1116) - [ST GET URL](#) (1288) - [ST INSERT EXPRESSION](#) (1281) - [ST INSERT URL](#) (1280) - [ST SET ATTRIBUTES](#) (1093) - [ST SET OPTIONS](#) (1289) - [ST SET PLAIN TEXT](#) (1136) - [ST SET TEXT](#) (1115) - [START MONITORING ACTIVITY](#) (1712) - [START SQL SERVER](#) (962) - [START TRANSACTION](#) (239) - [Std deviation](#) (26) - [STOP MONITORING ACTIVITY](#) (1721) - [STOP SQL SERVER](#) (963) - [Storage](#) (1525) - [String](#) (10) - [STRING LIST TO ARRAY](#) (511) - [Structure file](#) (489) - [Substring](#) (12) - [Subtotal](#) (97) - [Sum](#) (1) - [Sum squares](#) (28) - [Super](#) (1706) - [SUSPEND TRANSACTION](#) (1385) - [SVG EXPORT TO PICTURE](#) (1017) - [SVG Find element ID by coordinates](#) (1054) - [SVG Find element IDs by rect](#) (1109) - [SVG GET ATTRIBUTE](#) (1056) - [SVG SET ATTRIBUTE](#) (1055) - [SVG SHOW ELEMENT](#) (1108) - [System folder](#) (487) -

## T

[Table](#) (252) - [Table name](#) (256) - [Tan](#) (19) - [Temporary folder](#) (486) - [Test path name](#) (476) - [Test semaphore](#) (652) - [TEXT TO ARRAY](#) (1149) - [TEXT TO BLOB](#) (554) - [TEXT TO DOCUMENT](#) (1237) - [This](#) (1470) - [Tickcount](#) (458) - [Time](#) (179) - [Time string](#) (180) - [Timestamp](#) (1445) - [Tool bar height](#) (1016) - [TRACE](#) (157) - [Transaction level](#) (961) - [TRANSFORM PICTURE](#) (988) - [Trigger event](#) (369) - [Trigger level](#) (398) - [TRIGGER PROPERTIES](#) (399) - [True](#) (214) - [Trunc](#) (95) - [TRUNCATE TABLE](#) (1051) - [Type](#) (295) -

## U

[Undefined](#) (82) - [UNION](#) (120) - [UNLOAD RECORD](#) (212) - [UNREGISTER CLIENT](#) (649) - [Uppercase](#) (13) - [USE CHARACTER SET](#) (205) - [USE ENTITY SELECTION](#) (1513) - [USE NAMED SELECTION](#) (332) - [USE SET](#) (118) - [User in group](#) (338) - [USERS TO BLOB](#) (849) -

## V

[Validate password](#) (638) - [VALIDATE TRANSACTION](#) (240) - [Value type](#) (1509) - [VARIABLE TO BLOB](#) (532) - [VARIABLE TO VARIABLE](#) (635) - [Variance](#) (27) - [VERIFY CURRENT DATA FILE](#) (1008) - [VERIFY DATA FILE](#) (939) - [Verify password hash](#) (1534) - [Version type](#) (495) - [VOLUME ATTRIBUTES](#) (472) - [VOLUME LIST](#) (471) -

## W

[WA Back URL available](#) (1026) - [WA Create URL history menu](#) (1049) - [WA Evaluate JavaScript](#) (1029) - [WA EXECUTE JAVASCRIPT FUNCTION](#) (1043) - [WA Forward URL available](#) (1027) - [WA Get current URL](#) (1025) - [WA GET EXTERNAL LINKS FILTERS](#) (1033) - [WA Get last filtered URL](#) (1035) - [WA GET LAST URL ERROR](#) (1034) - [WA Get page content](#) (1038) - [WA Get page title](#) (1036) - [WA GET PREFERENCE](#) (1042) - [WA GET URL FILTERS](#) (1031) - [WA GET URL HISTORY](#) (1048) - [WA OPEN BACK URL](#) (1021) - [WA OPEN FORWARD URL](#) (1022) - [WA OPEN URL](#) (1020) - [WA OPEN WEB INSPECTOR](#) (1736) - [WA REFRESH CURRENT URL](#) (1023) - [WA Run offscreen area](#) (1727) - [WA SET EXTERNAL LINKS FILTERS](#) (1032) - [WA SET PAGE CONTENT](#) (1037) - [WA SET PREFERENCE](#) (1041) - [WA SET URL FILTERS](#) (1030) - [WA STOP LOADING URL](#) (1024) - [WA ZOOM IN](#) (1039) - [WA ZOOM OUT](#) (1040) - [WEB GET BODY PART](#) (1212) - [WEB Get body part count](#) (1211) - [WEB Get Current Session ID](#) (1162) -

[WEB GET HTTP BODY](#) (814) - [WEB GET HTTP HEADER](#) (697) - [WEB GET OPTION](#) (1209) - [WEB Get server info](#) (1531) - [WEB GET STATISTICS](#) (658) - [WEB GET VARIABLES](#) (683) - [WEB Is secured connection](#) (698) - [WEB Is server running](#) (1313) - [WEB LEGACY CLOSE SESSION](#) (1208) - [WEB LEGACY GET SESSION EXPIRATION](#) (1207) - [WEB SEND BLOB](#) (654) - [WEB SEND FILE](#) (619) - [WEB SEND HTTP REDIRECT](#) (659) - [WEB SEND RAW DATA](#) (815) - [WEB SEND TEXT](#) (677) - [WEB Server](#) (1674) - [WEB Server list](#) (1716) - [WEB SERVICE AUTHENTICATE](#) (786) - [WEB SERVICE CALL](#) (778) - [WEB SERVICE Get info](#) (780) - [WEB SERVICE GET RESULT](#) (779) - [WEB SERVICE SET OPTION](#) (901) - [WEB SERVICE SET PARAMETER](#) (777) - [WEB SET HOME PAGE](#) (639) - [WEB SET HTTP HEADER](#) (660) - [WEB SET OPTION](#) (1210) - [WEB SET ROOT FOLDER](#) (634) - [WEB START SERVER](#) (617) - [WEB STOP SERVER](#) (618) - [WEB Validate digest](#) (946) - [Window kind](#) (445) - [WINDOW LIST](#) (442) - [Window process](#) (446) - [Windows Alt down](#) (563) - [Windows Ctrl down](#) (562) - [WP Add picture](#) (1536) - [WP Bookmark range](#) (1416) - [WP COMPUTE FORMULAS](#) (1707) - [WP DELETE BOOKMARK](#) (1419) - [WP DELETE FOOTER](#) (1589) - [WP DELETE HEADER](#) (1588) - [WP DELETE PICTURE](#) (1701) - [WP DELETE STYLE SHEET](#) (1652) - [WP DELETE SUBSECTION](#) (1584) - [WP DELETE TEXT BOX](#) (1798) - [WP EXPORT DOCUMENT](#) (1337) - [WP EXPORT VARIABLE](#) (1319) - [WP Find all](#) (1755) - [WP Find next](#) (1764) - [WP Find previous](#) (1765) - [WP FREEZE FORMULAS](#) (1708) - [WP GET ATTRIBUTES](#) (1345) - [WP Get body](#) (1516) - [WP GET BOOKMARKS](#) (1417) - [WP Get breaks](#) (1768) - [WP Get data context](#) (1787) - [WP Get element by ID](#) (1549) - [WP Get elements](#) (1550) - [WP Get footer](#) (1504) - [WP Get formulas](#) (1702) - [WP Get frame](#) (1519) - [WP Get header](#) (1503) - [WP Get links](#) (1643) - [WP Get page count](#) (1412) - [WP Get position](#) (1577) - [WP Get section](#) (1581) - [WP Get sections](#) (1580) - [WP Get style sheet](#) (1656) - [WP Get style sheets](#) (1655) - [WP Get subsection](#) (1582) - [WP Get text](#) (1575) - [WP Get view properties](#) (1649) - [WP Import document](#) (1318) - [WP IMPORT STYLE SHEETS](#) (1673) - [WP INSERT BREAK](#) (1413) - [WP INSERT DOCUMENT](#) (1411) - [WP INSERT FORMULA](#) (1703) - [WP INSERT PICTURE](#) (1437) - [WP Insert table](#) (1473) - [WP Is font style supported](#) (1363) - [WP New](#) (1317) - [WP NEW BOOKMARK](#) (1415) - [WP New footer](#) (1587) - [WP New header](#) (1586) - [WP New style sheet](#) (1650) - [WP New subsection](#) (1583) - [WP New text box](#) (1797) - [WP Paragraph range](#) (1346) - [WP Picture range](#) (1347) - [WP PRINT](#) (1343) - [WP RESET ATTRIBUTES](#) (1344) - [WP SELECT](#) (1348) - [WP Selection range](#) (1340) - [WP SET ATTRIBUTES](#) (1342) - [WP SET DATA CONTEXT](#) (1786) - [WP SET FRAME](#) (1518) - [WP SET LINK](#) (1642) - [WP SET TEXT](#) (1574) - [WP SET VIEW PROPERTIES](#) (1648) - [WP Table append row](#) (1474) - [WP TABLE DELETE COLUMNS](#) (1694) - [WP TABLE DELETE ROWS](#) (1693) - [WP Table get cells](#) (1477) - [WP Table get columns](#) (1476) - [WP Table get rows](#) (1475) - [WP Table insert columns](#) (1692) - [WP Table insert rows](#) (1691) - [WP Table range](#) (1553) - [WP Text range](#) (1341) - [WP USE PAGE SETUP](#) (1358) - [WRITE PICTURE FILE](#) (680) -

## X

[XML DECODE](#) (1091) - [XML GET ERROR](#) (732) - [XML GET OPTIONS](#) (1096) - [XML SET OPTIONS](#) (1090) -

## Y

[Year of](#) (25) -

## Z

[ZIP Create archive](#) (1640) - [ZIP Read archive](#) (1637) -

## Control flow

Regardless of the simplicity or complexity of a method or function, you will always use one or more of three types of programming structures. Programming structures control the flow of execution, whether and in what order statements are executed within a method. There are three types of structures:

- **Sequential:** a sequential structure is a simple, linear structure. A sequence is a series of statements that 4D executes one after the other, from first to last. A one-line routine, frequently used for object methods, is the simplest case of a sequential structure. For example:  
`[People]lastName:=Uppercase([People]lastName)`
- **Branching:** A branching structure allows methods to test a condition and take alternative paths, depending on the result. The condition is a Boolean expression, an expression that evaluates TRUE or FALSE. One branching structure is the `If...Else...End if` structure, which directs program flow along one of two paths. The other branching structure is the `Case of...Else...End case` structure, which directs program flow to one of many paths.
- **Looping:** When writing methods, it is very common to find that you need a sequence of statements to repeat a number of times. To deal with this need, the 4D language provides the following looping structures:

- `While...End while`
- `Repeat...Until`
- `For...End for`
- `For each...End for each`

The loops are controlled in two ways: either they loop until a condition is met, or they loop a specified number of times. Each looping structure can be used in either way, but `While` loops and `Repeat` loops are more appropriate for repeating until a condition is met, and `For` loops are more appropriate for looping a specified number of times. `For each...End for each` allows mixing both ways and is designed to loop within objects and collections.

**Note:** 4D allows you to embed programming structures up to a "depth" of 512 levels.

### If...Else...End if

The formal syntax of the `If...Else...End if` control flow structure is:

```
If(Boolean_Expression)
    statement(s)
Else
    statement(s)
End if
```

Note that the `Else` part is optional; you can write:

```
If(Boolean_Expression)
    statement(s)
End if
```

The `If...Else...End if` structure lets your method choose between two actions, depending on whether a test (a Boolean expression) is TRUE or FALSE. When the

Boolean expression is TRUE, the statements immediately following the test are executed. If the Boolean expression is FALSE, the statements following the Else statement are executed. The Else statement is optional; if you omit Else, execution continues with the first statement (if any) following the End if.

Note that the Boolean expression is always fully evaluated. Consider in particular the following test:

```
If (MethodA & MethodB)
  ...
End if
```

The expression is TRUE only if both methods are TRUE. However, even if *MethodA* returns FALSE, 4D will still evaluate *MethodB*, which is a useless waste of time. In this case, it is more interesting to use a structure like:

```
If (MethodA)
  If (MethodB)
    ...
  End if
End if
```

The result is similar and *MethodB* is evaluated only if necessary.

**Note:** The [ternary operator](#) allows writing one-line conditional expressions and can replace a full sequence of If..Else statements.

## Example

```
// Ask the user to enter a name
$Find:=Request("Type a name")
If (OK=1)
  QUERY([People];[People]LastName=$Find)
Else
  ALERT("You did not enter a name.")
End if
```

**Tip:** Branching can be performed without statements to be executed in one case or the other. When developing an algorithm or a specialized application, nothing prevents you from writing:

```
If (Boolean_Expression)
Else
  statement(s)
End if
```

or:

```
If (Boolean_Expression)
  statement(s)
Else
End if
```

## Case of...Else...End case

The formal syntax of the Case of...Else...End case control flow structure is:

```

Case of
  :(Boolean_Expression)
    statement(s)
  :(Boolean_Expression)
    statement(s)
  .
  .
  .

  :(Boolean_Expression)
    statement(s)
Else
  statement(s)
End case

```

Note that the `Else` part is optional; you can write:

```

Case of
  :(Boolean_Expression)
    statement(s)
  :(Boolean_Expression)
    statement(s)
  .
  .
  .

  :(Boolean_Expression)
    statement(s)
End case

```

As with the `If...Else...End if` structure, the `Case of...Else...End case` structure also lets your method choose between alternative actions. Unlike the `If...Else...End if` structure, the `Case of...Else...End case` structure can test a reasonable unlimited number of Boolean expressions and take action depending on which one is TRUE.

Each Boolean expression is prefaced by a colon (`:`). This combination of the colon and the Boolean expression is called a case. For example, the following line is a case:

```
: (bValidate=1)
```

Only the statements following the first TRUE case (and up to the next case) will be executed. If none of the cases are TRUE, none of the statements will be executed (if no `Else` part is included).

You can include an `Else` statement after the last case. If all of the cases are FALSE, the statements following the `Else` will be executed.

## Example

This example tests a numeric variable and displays an alert box with a word in it:

```

Case of
  :(vResult=1) //Test if the number is 1
    ALERT("One.") //If it is 1, display an alert
  :(vResult=2) //Test if the number is 2
    ALERT("Two.") //If it is 2, display an alert
  :(vResult=3) //Test if the number is 3
    ALERT("Three.") //If it is 3, display an alert
Else //If it is not 1, 2, or 3, display an alert
  ALERT("It was not one, two, or three.")
End case

```

For comparison, here is the `If...Else...End if` version of the same method:

```
If(vResult=1) //Test if the number is 1
    ALERT("One.") //If it is 1, display an alert
Else
    If(vResult=2) //Test if the number is 2
        ALERT("Two.") //If it is 2, display an alert
    Else
        If(vResult=3) //Test if the number is 3
            ALERT("Three.") //If it is 3, display an alert
        Else //If it is not 1, 2, or 3, display an alert
            ALERT("It was not one, two, or three.")
        End if
    End if
End if
```

Remember that with a `Case of...Else...End case` structure, only the first TRUE case is executed. Even if two or more cases are TRUE, only the statements following the first TRUE case will be executed.

Consequently, when you want to implement hierarchical tests, you should make sure the condition statements that are lower in the hierarchical scheme appear first in the test sequence. For example, the test for the presence of condition1 covers the test for the presence of condition1&condition2 and should therefore be located last in the test sequence. For example, the following code will never see its last condition detected:

```
Case of
    :(vResult=1)
        ... //statement(s)
    :((vResult=1) & (vCondition#2)) //this case will never be detected
        ... //statement(s)
End case
```

In the code above, the presence of the second condition is not detected since the test "`vResult=1`" branches off the code before any further testing. For the code to operate properly, you can write it as follows:

```
Case of
    :((vResult=1) & (vCondition#2)) //this case will be detected first
        ... //statement(s)
    :(vResult=1)
        ... //statement(s)
End case
```

Also, if you want to implement hierarchical testing, you may consider using hierarchical code.

**Tip:** Branching can be performed without statements to be executed in one case or another. When developing an algorithm or a specialized application, nothing prevents you from writing:

```
Case of
    :(Boolean_Expression)
    :(Boolean_Expression)
    ...
    :(Boolean_Expression)
        statement(s)
Else
    statement(s)
End case
```

or:

```

Case of
  : (Boolean_Expression)
  : (Boolean_Expression)
    statement(s)
    ...
    : (Boolean_Expression)
      statement(s)
  Else
End case

```

or:

```

Case of
  Else
    statement(s)
End case

```

## While...End while

The formal syntax of the `While...End while` control flow structure is:

```

While(Boolean_Expression)
  statement(s)
  {break}
  {continue}
End while

```

A `While...End while` loop executes the statements inside the loop as long as the Boolean expression is TRUE. It tests the Boolean expression at the beginning of the loop and does not enter the loop at all if the expression is FALSE.

The `break` and `continue` statements are [described below](#).

It is common to initialize the value tested in the Boolean expression immediately before entering the `While...End while` loop. Initializing the value means setting it to something appropriate, usually so that the Boolean expression will be TRUE and `While...End while` executes the loop.

The Boolean expression must be set by something inside the loop or else the loop will continue forever. The following loop continues forever because `NeverStop` is always TRUE:

```

NeverStop:=True
While(NeverStop)
End while

```

If you find yourself in such a situation, where a method is executing uncontrolled, you can use the trace facilities to stop the loop and track down the problem. For more information about tracing a method, see the [Error handling](#) page.

### Example

```

CONFIRM("Add a new record?") //The user wants to add a record?
While(OK=1) //Loop as long as the user wants to
  ADD RECORD([aTable]) //Add a new record
End while //The loop always ends with End while

```

In this example, the `OK` system variable is set by the `CONFIRM` command before the loop starts. If the user clicks the **OK** button in the confirmation dialog box, the `OK` system variable is set to 1 and the loop starts. Otherwise, the `OK` system variable is

set to 0 and the loop is skipped. Once the loop starts, the `ADD RECORD` command keeps the loop going because it sets the `OK` system variable to 1 when the user saves the record. When the user cancels (does not save) the last record, the `OK` system variable is set to 0 and the loop stops.

## Repeat...Until

The formal syntax of the `Repeat...Until` control flow structure is:

```
Repeat
    statement(s)
    {break}
    {continue}
Until(Boolean_Expression)
```

A `Repeat...Until` loop is similar to a `While...End while` loop, except that it tests the Boolean expression after the loop rather than before. Thus, a `Repeat...Until` loop always executes the loop once, whereas if the Boolean expression is initially False, a `While...End while` loop does not execute the loop at all.

The other difference with a `Repeat...Until` loop is that the loop continues until the Boolean expression is TRUE.

The `break` and `continue` statements are [described below](#).

### Example

Compare the following example with the example for the `While...End while` loop. Note that the Boolean expression does not need to be initialized—there is no `CONFIRM` command to initialize the `OK` variable.

```
Repeat
    ADD RECORD([aTable])
Until(OK=0)
```

## For...End for

The formal syntax of the `For...End for` control flow structure is:

```
For(Counter_Variable;Start_Expression;End_Expression{;Increment_Express
    statement(s)
    {break}
    {continue}
End for
```

The `For...End for` loop is a loop controlled by a counter variable:

- The counter variable `Counter_Variable` is a numeric variable (Real or Long Integer) that the `For...End for` loop initializes to the value specified by `Start_Expression`.
- Each time the loop is executed, the counter variable is incremented by the value specified in the optional value `Increment_Expression`. If you do not specify `Increment_Expression`, the counter variable is incremented by one (1), which is the default.
- When the counter variable passes the `End_Expression` value, the loop stops.

**Important:** The numeric expressions `Start_Expression`, `End_Expression` and `Increment_Expression` are evaluated once at the beginning of the loop. If these

expressions are variables, changing one of these variables within the loop will not affect the loop.

**Tip:** However, for special purposes, you can change the value of the counter variable *Counter\_Variable* within the loop; this will affect the loop.

- Usually *Start\_Expression* is less than *End\_Expression*.
- If *Start\_Expression* and *End\_Expression* are equal, the loop will execute only once.
- If *Start\_Expression* is greater than *End\_Expression*, the loop will not execute at all unless you specify a negative *Increment\_Expression*. See the examples.

The `break` and `continue` statements are [described below](#).

## Basic examples

1. The following example executes 100 iterations:

```
For(vCounter;1;100)
  //Do something
End for
```

2. The following example goes through all elements of the array *anArray*:

```
For($v1Elem;1;Size of array(anArray))
  //Do something with the element
  anArray{$v1Elem}:=...
End for
```

3. The following example goes through all the characters of the text *vtSomeText*:

```
For($v1Char;1;Length(vtSomeText))
  //Do something with the character if it is a TAB
  If(Character code(vtSomeText[[${v1Char}]])=Tab)
  //...
  End if
End for
```

4. The following example goes through the selected records for the table [*aTable*]:

```
FIRST RECORD([aTable])
For($v1Record;1;Records in selection([aTable]))
  //Do something with the record
  SEND RECORD([aTable])
  //...
  //Go to the next record
  NEXT RECORD([aTable])
End for
```

Most of the `For...End for` loops you will write in your projects will look like the ones listed in these examples.

## Counter variable

### Decrementing counter variable

In some cases, you may want to have a loop whose counter variable is decreasing rather than increasing. To do so, you must specify *Start\_Expression* greater than *End\_Expression* and a negative *Increment\_Expression*. The following examples do the same thing as the previous examples, but in reverse order:

5. The following example executes 100 iterations:

```
For (vCounter;100;1;-1)
  //Do something
End for
```

6. The following example goes through all elements of the array anArray:

```
For ($vlElem;Size of array(anArray);1;-1)
  //Do something with the element
  anArray{$vlElem}:=...
End for
```

7. The following example goes through all the characters of the text vtSomeText:

```
For ($vlChar;Length(vtSomeText);1;-1)
  //Do something with the character if it is a TAB
  If(Character code(vtSomeText[[${vlChar}]])=Tab)
  //...
  End if
End for
```

8. The following example goes through the selected records for the table [aTable]:

```
LAST RECORD([aTable])
For ($vlRecord;Records in selection([aTable]);1;-1)
  //Do something with the record
  SEND RECORD([aTable])
  //...
  //Go to the previous record
  PREVIOUS RECORD([aTable])
End for
```

### **Incrementing the counter variable by more than one**

If you need to, you can use an *Increment\_Expression* (positive or negative) whose absolute value is greater than one.

9. The following loop addresses only the even elements of the array anArray:

```
For ($vlElem;2;Size of array(anArray);2)
  //Do something with the element #2,#4...#2n
  anArray{$vlElem}:=...
End for
```

### **Optimizing the execution of the For...End for loops**

You can use Real and Integer variables as well as interprocess, process, and local variable counters. For lengthy repetitive loops, especially in compiled mode, use local Long Integer variables.

10. Here is an example:

```
var $vlCounter : Integer //use local Integer variables
For ($vlCounter;1;10000)
  //Do something
End for
```

### **Comparing looping structures**

Let's go back to the first `For...End for` example. The following example executes 100 iterations:

```
For(vCounter;1;100)
  //Do something
End for
```

It is interesting to see how the `While...End while` loop and `Repeat...Until` loop would perform the same action. Here is the equivalent `While...End while` loop:

```
$i:=1 //Initialize the counter
While($i<=100) //Loop 100 times
  //Do something
  $i:=$i+1 //Need to increment the counter
End while
```

Here is the equivalent `Repeat...Until` loop:

```
$i:=1 //Initialize the counter
Repeat
  //Do something
  $i:=$i+1 //Need to increment the counter
Until($i=100) //Loop 100 times
```



The `For...End for` loop is usually faster than the `While...End while` and `Repeat...Until` loops, because 4D tests the condition internally for each cycle of the loop and increments the counter. Therefore, use the `For...End for` loop whenever possible.

## Nested For...End for looping structures

You can nest as many control structures as you (reasonably) need. This includes nesting `For...End for` loops. To avoid mistakes, make sure to use different counter variables for each looping structure.

Here are two examples:

1. The following example goes through all the elements of a two-dimensional array:

```
For($v1Elem;1;Size of array(anArray))
  //...
  //Do something with the row
  //...
  For($v1SubElem;1;Size of array(anArray{$v1Elem}))
    //Do something with the element
    anArray{$v1Elem}{$v1SubElem}:=...
  End for
End for
```

2. The following example builds an array of pointers to all the date fields present in the database:

```

ARRAY POINTER($apDateFields;0)
$vlElem:=0
For($vlTable;1;Get last table number)
    If(Is table number valid($vlTable))
        For($vlField;1;Get last field number($vlTable))
            If(Is field number valid($vlTable;$vlField))
                $vpField:=Field($vlTable;$vlField)
                If(Type($vpField->)=Is date)
                    $vlElem:=$vlElem+1
                    INSERT IN ARRAY($apDateFields;$vlElem)
                    $apDateFields{$vlElem}:=$vpField
                End if
            End if
        End for
    End if
End for

```

## For each...End for each

The formal syntax of the `For each...End for each` control flow structure is:

```

For each(Current_Item;Expression{;begin{;}end{}}){Until|While}
(Boolean_Expression)
    statement(s)
    {break}
    {continue}
End for each

```

The `For each...End for each` structure iterates a specified *Current\_item* over all values of the *Expression*. The *Current\_item* type depends on the *Expression* type. The `For each...End for each` loop can iterate through three *Expression* types:

- collections: loop through each element of the collection,
- entity selections: loop through each entity,
- objects: loop through each object property.

The following table compares the three types of `For each...End for each`:

|                                   | <b>Loop through collections</b>                  | <b>Loop through entity selections</b> | <b>Loop through objects</b> |
|-----------------------------------|--|---------------------------------------|-----------------------------|
| Current_Item type                 | Variable of the same type as collection elements | Entity                                | Text variable               |
| Expression type                   | Collection (with elements of the same type)      | Entity selection                      | Object                      |
| Number of loops (by default)      | Number of collection elements                    | Number of entities in the selection   | Number of object properties |
| Support of begin / end parameters | Yes  | Yes                                   | No                          |

- The number of loops is evaluated at startup and will not change during the processing. Adding or removing items during the loop is usually not recommended since it may result in missing or redundant iterations.
- By default, the enclosed *statement(s)* are executed for each value in *Expression*. It is, however, possible to exit the loop by testing a condition either at the beginning of the loop (`While`) or at the end of the loop (`Until`).
- The *begin* and *end* optional parameters can be used with collections and entity selections to define boundaries for the loop.
- The `For each...End for each` loop can be used on a **shared collection** or a **shared object**. If your code needs to modify one or more element(s) of the

collection or object properties, you need to use the `Use...End use` keywords. Depending on your needs, you can call the `Use...End use` keywords:

- before entering the loop, if items should be modified together for integrity reasons, or
- within the loop when only some elements/properties need to be modified and no integrity management is required.

The `break` and `continue` statements are [described below](#).

## Loop through collections

When `For each...End for each` is used with an *Expression* of the *Collection* type, the *Current\_Item* parameter is a variable of the same type as the collection elements. By default, the number of loops is based on the number of items of the collection.

The collection must contain only elements of the same type, otherwise an error will be returned as soon as the *Current\_Item* variable is assigned the first mismatched value type.

At each loop iteration, the *Current\_Item* variable is automatically filled with the matching element of the collection. The following points must be taken into account:

- If the *Current\_Item* variable is of the object type or collection type (i.e. if *Expression* is a collection of objects or of collections), modifying this variable will automatically modify the matching element of the collection (because objects and collections share the same references). If the variable is of a scalar type, only the variable will be modified.
- The *Current\_Item* variable must be of the same type as the collection elements. If any collection item is not of the same type as the variable, an error is generated and the loop stops.
- If the collection contains elements with a **Null** value, an error will be generated if the *Current\_Item* variable type does not support **Null** values (such as longint variables).

## Example

You want to compute some statistics for a collection of numbers:

```
var $nums : Collection
$nums:=New collection(10;5001;6665;33;1;42;7850)
var $item;$vEven;$vOdd;$vUnder;$vOver : Integer
For each($item;$nums)
  If($item%2=0)
    $vEven:=$vEven+1
  Else
    $vOdd:=$vOdd+1
  End if
  Case of
    : ($item<5000)
      $vUnder:=$vUnder+1
    : ($item>6000)
      $vOver:=$vOver+1
  End case
End for each
//$vEven=3, $vOdd=4
//$vUnder=4, $vOver=2
```

## Loop through entity selections

When `For each...End for each` is used with an *Expression* of the *Entity selection*

type, the *Current\_Item* parameter is the entity that is currently processed.

The number of loops is based on the number of entities in the entity selection. On each loop iteration, the *Current\_Item* parameter is automatically filled with the entity of the entity selection that is currently processed.

**Note:** If the entity selection contains an entity that was removed meanwhile by another process, it is automatically skipped during the loop.

Keep in mind that any modifications applied on the current entity must be saved explicitly using `entity.save()`.

## Example

You want to raise the salary of all British employees in an entity selection:

```
var emp : Object
For each(emp;ds.Employees.query("country='UK'"))
    emp.salary:=emp.salary*1,03
    emp.save()
End for each
```

## Loop through object properties

When `For each...End for each` is used with an *Expression* of the Object type, the *Current\_Item* parameter is a text variable automatically filled with the name of the currently processed property.

The properties of the object are processed according to their order of creation. During the loop, properties can be added to or removed from the object, without modifying the number of loops that will remain based on the original number of properties of the object.

## Example

You want to switch the names to uppercase in the following object:

```
{
    "firstname": "gregory",
    "lastname": "badikora",
    "age": 20
}
```

You can write:

```
For each(property;vObject)
    If(Value type(vObject[property])=Is text)
        vObject[property]:=Uppercase(vObject[property])
    End if
End for each
{
    "firstname": "GREGORY",
    "lastname": "BADIKORA",
    "age": 20
}
```

## begin / end parameters

You can define bounds to the iteration using the optional begin and end parameters.

**Note:** The *begin* and *end* parameters can only be used in iterations through collections and entity selections (they are ignored on object properties).

- In the *begin* parameter, pass the element position in *Expression* at which to start the iteration (*begin* is included).
- In the *end* parameter, you can also pass the element position in *Expression* at which to stop the iteration (*end* is excluded).

If *end* is omitted or if *end* is greater than the number of elements in *Expression*, elements are iterated from *begin* until the last one (included). If the *begin* and *end* parameters are positive values, they represent actual positions of elements in *Expression*. If *begin* is a negative value, it is recalculated as `begin:=begin+Expression size` (it is considered as the offset from the end of *Expression*). If the calculated value is negative, *begin* is set to 0. **Note:** Even if *begin* is negative, the iteration is still performed in the standard order. If *end* is a negative value, it is recalculated as `end:=end+Expression size`

For example:

- a collection contains 10 elements (numbered from 0 to 9)
- *begin*=-4 -> *begin*=-4+10=6 -> iteration starts at the 6th element (#5)
- *end*=-2 -> *end*=-2+10=8 -> iteration stops before the 8th element (#7), i.e. at the 7th element.

## Example

```
var $col;$col2 : Collection
$col:=New collection("a";"b";"c";"d";"e")
$col2:=New collection(1;2;3)
var $item : Text
For each($item;$col;0;3)
    $col2.push($item)
End for each
//$col2=[1,2,3,"a","b","c"]
For each($item;$col;-2;-1)
    $col2.push($item)
End for each
//$col2=[1,2,3,"a","b","c","d"]
```

## Until and While conditions

You can control the `For each...End for each` execution by adding an `Until` or a `While` condition to the loop. When an `Until(condition)` statement is associated to the loop, the iteration will stop as soon as the condition is evaluated to `True`, whereas when is case of a `While(condition)` statement, the iteration will stop when the condition is first evaluated to `False`.

You can pass either keyword depending on your needs:

- The `Until` condition is tested at the end of each iteration, so if the *Expression* is not empty or null, the loop will be executed at least once.
- The `While` condition is tested at the beginning of each iteration, so according to the condition result, the loop may not be executed at all.

## Example

```

$colNum:=New collection(1;2;3;4;5;6;7;8;9;10)

$total:=0
For each($num;$colNum)While($total<30) //tested at the beginning
    $total:=$total+$num
End for each
ALERT(String($total)) // $total = 36 (1+2+3+4+5+6+7+8)

$total:=1000
For each($num;$colNum)Until($total>30) //tested at the end
    $total:=$total+$num
End for each
ALERT(String($total)) // $total = 1001 (1000+1)

```

## break and continue

All looping structures above support both `break` and `continue` statements. These statements give you more control over the loops by allowing to exit the loop and to bypass the current iteration at any moment.

### break

The `break` statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop.

If the `break` statement is inside a [nested loop](#) (loop inside another loop), the `break` statement will terminate the innermost loop.

### Example

```

For (vCounter;1;100)
    If ($stab{vCounter}!="") //if a condition becomes true
        break //end of the for loop
    End if
End for

```

### continue

The `continue` statement terminates execution of the statements in the current iteration of the current loop, and continues execution of the loop with the next iteration.

```

var $text : Text
For ($i; 0; 9)
    If ($i=3)
        continue //go directly to the next iteration
    End if
    $text:=$text+String($i)
End for
// $text="012456789"

```

## return {expression}

- History

The `return` statement can be called from anywhere. When a `return` statement is used in a function or method, the execution of the function or method is stopped. The remaining code is not executed and the control is returned to the caller.

The `return` statement can be used to [return a value](#) to the caller.

## Example

```
var $message : Text
var $i : Integer

While (True) //infinite loop
    $i:=$i+1
    $message+=String($i)+"A\r" // until 5
    logConsole($message)
    If ($i=5)
        return //stops the loop
    End if
    $message+=String($i)+"B\r" // until 4
    logConsole($message)
End while
$message+=String($i)+"C\r" //never executed
logConsole($message)

// 1A
// 1B
// 2A
// 2B
// 3A
// 3B
// 4A
// 4B
// 5A
```

## Error handling

Error handling is the process of anticipating and responding to errors that might occur in your application. 4D provides a comprehensive support for catching and reporting errors at runtime, as well as for investigating their conditions.

Error handling meets two main needs:

- finding out and fixing potential errors and bugs in your code during the development phase,
- catching and recovering from unexpected errors in deployed applications; in particular, you can replace system error dialogs (disk full, missing file, etc.) with your own interface.



### GOOD PRACTICE

It is highly recommended to install a global error-handling method on 4D Server, for all code running on the server. When 4D Server is not running [headless](#) (i.e. launched with its [administration window](#)), this method would avoid unexpected dialog boxes to be displayed on the server machine. In headless mode, errors are logged in the [4DDebugLog file](#) for further analysis.

## Error or status

Many 4D class functions, such as `entity.save()` or `transporter.send()`, return a *status* object. This object is used to store "predictable" errors in the runtime context, e.g. invalid password, locked entity, etc., that do not stop program execution. This category of errors can be handled by regular code.

Other "unpredictable" errors include disk write error, network failure, or in general any unexpected interruption. This category of errors generates exceptions and needs to be handled through an error-handling method.

## Installing an error-handling method

In 4D, all errors can be caught and handled by specific project methods, named **error-handling** (or **error-catching**) methods.

Once installed, error handlers are automatically called in interpreted or compiled mode in case of error in the 4D application or one of its components. A different error handler can be called depending on the execution context (see below).

To *install* an error-handling project method, you just need to call the [ON\\_ERR\\_CALL](#) command with the project method name and (optionnally) scope as parameters. For example:

```
ON ERR CALL("IO_Errors";ek local) //Installs a local error-handling method
```

To stop catching errors for an execution context and give back hand, call [ON\\_ERR\\_CALL](#) with an empty string:

```
ON ERR CALL("");ek local) //gives back control for the local process
```

The [Method called on error](#) command allows you to know the name of the method installed by [ON\\_ERR\\_CALL](#) for the current process. It is particularly useful in

the context of generic code because it enables you to temporarily change and then restore the error-catching method:

```
$methCurrent:=Method called on error(ek local)
ON ERR CALL("NewMethod";ek local)
//If the document cannot be opened, an error is generated
$ref:=Open document("MyDocument")
//Reinstallation of previous method
ON ERR CALL($methCurrent;ek local)
```

## Scope and components

An error-handling method can be set for different execution contexts:

- for the **current process**- a local error handler will be only called for errors that occurred in the current process of the current project,
- for the **whole application**- a global error handler will be called for all errors that occurred in the application execution context of the current project,
- from the **components**- this error handler is defined in a host project and will be called for all errors that occurred in the components when they were not already caught by a component handler.

Examples:

```
ON ERR CALL("IO_Errors";ek local) //Installs a local error-handling
method
ON ERR CALL("globalHandler";ek global) //Installs a global error-
handling method
ON ERR CALL("componentHandler";ek errors from components) //Installs an
error-handling method for components
```

You can install a global error handler that will serve as "fallback" and specific local error handlers for certain processes. A global error handler is also useful on the server to avoid error dialogs on the server when run with interface.

You can define a single error-catching method for the whole application or different methods per application module. However, only one method can be installed per execution context and per project.

When an error occurs, only one method is called, as described in the following diagram:

## Handling errors within the method

Within a custom error method, you have access to several pieces of information that will help you identifying the error:

- dedicated system variables:
  - `Error` (`longint`): error code
  - `Error method` (`text`): name of the method that triggered the error
  - `Error line` (`longint`): line number in the method that triggered the error
  - `Error formula` (`text`): formula of the 4D code (raw text) which is at the origin of the error.

4D automatically maintains a number of variables called **system variables**, meeting different needs. See the *4D Language Reference manual*.

- the [Last errors](#) command that returns a collection of the current stack of errors that occurred in the 4D application. You can also use the [GET LAST ERROR STACK](#) command that returns the same information as arrays.
- the [Get call chain](#) command that returns a collection of objects describing each step of the method call chain within the current process.

## Example

Here is a simple error-handling system:

```
//installing the error handling method
ON ERR CALL("errorMethod")
//... executing code
ON ERR CALL("") //giving control back to 4D
// errorMethod project method
If(Error#1006) //this is not a user interruption
    ALERT("The error "+String(Error)+" occurred". The code in question
is: \"""+Error formula+"\"")
End if
```

## Using an empty error-handling method

If you mainly want the standard error dialog box to be hidden, you can install an empty error-handling method. The [Error](#) system variable can be tested in any method, i.e. outside of the error-handling method:

```
ON ERR CALL("emptyMethod") //emptyMethod exists but is empty
$doc:=Open document( "myFile.txt")
If (Error=-43)
    ALERT("File not found.")
End if
ON ERR CALL("")
```

## Interpreted and Compiled modes

4D applications can work in **interpreted** or **compiled** mode:

- in interpreted mode, statements are read and translated in machine language at the moment of their execution. You can add or modify the code whenever you need to, the application is automatically updated.
- in compiled mode, all methods are read and translated once, at the compilation step. Afterwards, the application only contains assembly level instructions are available, it is no longer possible to edit the code.

The advantages of the compilation are:

- **Speed:** Your application can run from 3 to 1,000 times faster.
- **Code checking:** Your application is scanned for the consistency of code. Both logical and syntactical conflicts are detected.
- **Protection:** Once your application is compiled, you can delete the interpreted code. Then, the compiled application is functionally identical to the original, except that the structure and methods cannot be viewed or modified, deliberately or inadvertently.
- **Stand-alone double-clickable applications:** compiled applications can also be transformed into stand-alone applications with their own icon.
- **Preemptive mode:** only compiled code can be executed in preemptive processes.

## Differences between interpreted and compiled code

Although application will work the same way in interpreted and compiled modes, there are some differences to know when you write code that will be compiled. The 4D interpreter is usually more flexible than the compiler.

| Compiled   | Interpreted  |
|--|--|
| You cannot have a method with the same name as a variable.   | No error is generated, but priority is given to the method                     |
| All variables must be typed, either through a declaration (using <code>var</code> , <code>#Declare</code> , or <code>Function</code> keywords), or by the compiler at compilation time.    | Variables can be typed on-the-fly (not recommended)                            |
| You cannot change the data type of any variable or array.  | Changing the data type of a variable or an array is possible (not recommended) |
| You cannot change a one-dimensional array to a two-dimensional array, or change a two-dimensional array to a one-dimensional array.  | Possible   |
| Although the compiler will type the variable for you, you should specify the data type of a variable by using declarations where the data type is ambiguous, such as in a form.            |  |
| The <code>Undefined</code> function always returns <code>False</code> for variables.   |  |
| Variables are always defined.  |  |
| If you have checked the "Can be run in preemptive processes" property for the method, the code must not call Preemptive process any thread-unsafe commands or other thread-unsafe methods. | properties are ignored   |
| The <code>IDLE</code> command is necessary to call 4D in specific loops  | It is always possible to interrupt 4D  |

## Using Compiler Directives with the Interpreter

Compiler directives are not required for uncompiled applications. The interpreter automatically types each variable according to how it is used in each statement, and a variable can be freely retyped throughout the application project.

Because of this flexibility, it is possible that an application can perform differently in interpreted and compiled modes.

For example, if you write:

```
var MyInt : Integer
```

and elsewhere in the project, you write:

```
MyInt:=3.1416
```

In this example, `MyInt` is assigned the same value (3) in both the interpreted and compiled modes, provided the compiler directive is interpreted *prior* to the assignment statement.

The 4D interpreter uses compiler directives to type variables. When the interpreter encounters a compiler directive, it types the variable according to the directive. If a subsequent statement tries to assign an incorrect value (e.g., assigning an alphanumeric value to a numeric variable) the assignment will not take place and will generate an error.

The order in which the two statements appear is irrelevant to the compiler, because it first scans the entire project for compiler directives. The interpreter, however, is not systematic. It interprets statements in the order in which they are executed. That

order, of course, can change from session to session, depending on what the user does. For this reason, it is important to design your project so that your compiler directives are executed prior to any statements containing declared variables.

## Using pointers to avoid retyping

A variable cannot be retyped. However, it is possible to use a pointer to refer to variables of different data types. For example, the following code is allowed in both interpreted and compiled modes:

```
var $p : Pointer
var $name : Text
var $age : Integer

$name:="Smith"
$age:=50

$p:=->$name //text target for the pointer
$p->:="Wesson" //assigns a text value

$p:=->$age
// new target of different type for the pointer
$p->:=55 //assigns a number value
```

Imagine a function that returns the length (number of characters) of values that can be of any type.

```
// Calc_Length (how many characters)
// $vp = pointer to flexible variable type, numeric, text, time,
boolean

#DECLARE($vp : Pointer) -> $length : Integer
var $result : Text
$result:=String($vp->)
$length:=Length($result)
```

Then this method can be called:

```
$var1:="my text"
$var2:=5.3
$var3:=?10:02:24?
$var4:=True

$vLength:=Calc_Length(->$var1)+Calc_Length(->$var2)+Calc_Length (-
>$var3)+Calc_Length(->$var4)

ALERT("Total length: "+String($vLength))
```

# Components

A 4D component is a set of 4D code and forms representing one or more functionalities that you can install and use in your projects. For example, the [4D SVG component](#) adds advanced commands and an integrated rendering engine that can be used to display SVG files.

## Where to find components?

Several components are [preinstalled in the 4D development environment](#), but a lot of 4D components from the 4D community [can be found on GitHub](#). Additionally, you can [develop your own 4D components](#).

## Installing components

To install a component, you simply need to copy the component files into the [Components folder of the project](#). You can use aliases or shortcuts.

A host project running in interpreted mode can use either interpreted or compiled components. A host project running in compiled mode cannot use interpreted components. In this case, only compiled components can be used.

## Using components

Exposed component code (methods and functions) as well as forms can be used as standard elements in your 4D development.

When an installed component contains methods, classes, and functions, they appear in the **Component Methods** theme of the Explorer's Methods page:

### NOTE

If the component is compiled, its [namespace](#) is written between parentheses after its name. Use this namespace to access the component's functions.

You can select a component [project method](#) or [class](#) and click on the **Documentation** button of the Explorer to get information about it, [if any](#).

## Plug-ins

As you develop a 4D application, you will discover many capabilities that you did not notice when you started. You can even augment the standard version of 4D by adding **plug-ins** to your 4D development environment.

### What is a plug-in and what can it do?

A plug-in is a piece of code, written in any language such as C or C++, that 4D launches at start up. It adds functionality to 4D and thus increases its capacity. A plug-in usually contains a set of routines given to the 4D developer. It can handle external areas and run external processes.

### Where to find plug-ins?

Multiple plug-ins have already been written by the 4D community. Published plug-ins [can be found on GitHub](#). Additionnally, you can [develop your own plug-ins](#).

### Installing plug-ins

You install plug-ins in the 4D environment by copying their files into the **Plugins** folder, at the [same level as the Project folder](#).

Plug-ins are loaded by 4D when the application is launched so you will need to quit your 4D application before installing them. If a plug-in requires a specific license for use, it will be loaded but not available for use.

### Using plug-ins

Plug-ins commands can be used as regular 4D commands in your 4D development. Plug-in commands appear in the **Plug-ins** page of the Explorer.

## Identifiers

This section describes the conventions and rules for naming various elements in the 4D language (variables, object properties, tables, forms, etc.).

If non-Roman characters are used in the names of the identifiers, their maximum length may be smaller.

## Arrays

Array names follow the same rules as [variables](#).

## Classes

The name of a class can contain up to 31 characters.

A class name must be compliant with standard [property naming rules](#) for dot notation.

Giving the same name to a class and a [database table](#) is not recommended, in order to prevent any conflict.

## Functions

Function names must be compliant with standard [property naming rules](#) for dot notation.

**Tip:** Starting the function name with an underscore character ("\_") will exclude the function from the autocompletion features in the 4D code editor.

## Object properties

The name of an object property (also called object *attribute*) can contain up to 255 characters.

Object properties can reference scalar values, ORDA elements, class functions, other objects, etc. Whatever their nature, object property names must follow the following rules **if you want to use the [dot notation](#):**

- A property name must begin with a letter, an underscore, or a dollar "\$".
- Thereafter, the name can include any letter, digit, the underscore character ("\_"), or the dollar character ("\$").
- Property names are case sensitive.

Examples:

```
myObject.myAttribute:="10"  
$value:=$clientObj.data.address.city
```



Starting an object property name with an underscore character ("\_") will exclude the property from the autocompletion features in the 4D code editor. For example, if you declare `$o._myPrivateProperty`, it will not be proposed in the code editor when you type in `"$o. "`.

See also [ECMA Script standard](#).

## NOTE

If you use **string notation** within square brackets, property names can contain any characters (ex: `myObject["1. First property"]`).

## Parameters

Parameter names must start with a `$` character and follow the same rules as [variable names](#).

Examples:

```
Function getArea($width : Integer; $height : Integer) -> $area : Integer  
#DECLARE ($i : Integer ; $param : Date) -> $myResult : Object
```

## Project methods

The name of a project method name contain up to 31 characters.

- A project method name must begin with a letter, a digit, or an underscore
- Thereafter, the name can include any letter or digit, the underscore character ("`_`"), or the space character.
- Do not use reserved names, i.e. 4D command names (`Date`, `Time`, etc), keywords (`If`, `For`, etc.), or constant names (`Euro`, `Black`, `Friday`, etc.).
- Project method names are case insensitive.

Examples:

```
If (New client)  
DELETE DUPLICATED VALUES  
APPLY TO SELECTION([Employees]; INCREASE SALARIES)
```

**Tip:** It is a good programming technique to adopt the same naming convention as the one used by 4D for built-in methods. Use uppercase characters for naming your methods; however if a method returns a value, capitalize the first character of its name. By doing so, when you reopen a project for maintenance after a few months, you will already know if a method returns a result by simply looking at its name in the Explorer window.

When you call a method, you just type its name. However, some 4D built-in commands, such as `ON EVENT CALL`, as well as all plug-in commands, expect the name of a method as a string when a method parameter is passed.

Examples:

```
//This command expects a method (function) or formula  
QUERY BY FORMULA([aTable];Special query)  
//This command expects a method (procedure) or statement  
APPLY TO SELECTION([Employees]; INCREASE SALARIES)  
//But this command expects a method name  
ON EVENT CALL("HANDLE EVENTS")
```

## Tables and Fields

You designate a table by placing its name between brackets: [...]. You designate a field by first specifying the table to which it belongs (the field name immediately follows the table name).

A table name and field name can contain up to 31 characters.

- A table or field name must begin with a letter, an underscore, or a dollar ("\$")
- Thereafter, the name can include alphabetic characters, numeric characters, the space character, and the underscore character ("\_").
- Do not use reserved names, i.e. 4D command names (Date, Time, etc), keywords (If, For, etc.), or constant names (Euro, Black, Friday, etc.).
- Additional rules must be respected when the database must be handled via SQL: only the characters \_0123456789abcdefghijklmnopqrstuvwxyz are accepted, and the name must not include any SQL keywords (command, attribute, etc.).

Examples:

```
FORM SET INPUT([Clients];"Entry")
ADD RECORD([Letters])
[Orders]Total:=Sum([Line]Amount)
QUERY([Clients];[Clients]Name="Smith")
[Letters]Text:=Capitalize text([Letters]Text)
```

Giving the same name to a table and a [class](#) is not recommended, in order to prevent any conflict.

## Variables

The name of a variable can be up to 31 characters, not including the scope symbols (\$ or <>).

- A variable name must begin with a letter, an underscore, or a dollar ("\$") for [parameters](#) and [local variables](#), or <> for [interprocess variables](#).
- A digit as first character is allowed but not recommended, and is not supported by the [var declaration syntax](#).
- Thereafter, the name can include any letter or digit, and the underscore character ("\_").
- Space character is allowed but not recommended, and is not supported by the [var declaration syntax](#).
- Do not use reserved names, i.e. 4D command names (Date, Time, etc), keywords (If, For, etc.), or constant names (Euro, Black, Friday, etc.).
- Variable names are case insensitive.

Examples:

```
For($vlRecord;1;100) //local variable
$vsMyString:="Hello there" //local variable
var $vName; $vJob : Text //local variales
If(bValidate=1) //process variable
<>vlProcessID:=Current process() //interprocess variable
```

## Other names

In the 4D language, several elements have their names handled as strings: **forms**, **form objects**, **named selections**, **processes**, **sets**, **menu bars**, etc.

Such string names can contain up to 255 characters, not including the \$ or <> characters (if any).

- String names can contain any characters.
- String names are case insensitive.

## Examples:

```
DIALOG([Storage];"Note box"+String($vlStage))
OBJECT SET FONT(*;"Binfo";"Times")
USE NAMED SELECTION([Customers];"Closed")//Process Named Selection
USE NAMED SELECTION([Customers];"<>ByZipcode") //Interprocess Named Selection
    //Starting the global process "Add Customers"
$vlProcessID:=New process("P_ADD_CUSTOMERS";48*1024;"Add Customers")
    //Starting the local process "$Follow Mouse Moves"
$vlProcessID:=New process("P_MOUSE_SNIFFER";16*1024;"$Follow Mouse Moves")
CREATE SET([Customers];"Customer Orders")//Process set
USE SET("<>Deleted Records") //Interprocess set
If(Records in set("$Selection"+String($i))>0) //Client set
```

## Pathnames

File and Folder functions, properties, and commands allow you to handle files and folders as objects. This makes file and folder management powerful and flexible. For example, to create a new file in the current user's Documents folder, you can write:

```
$ok:=Folder(fk documents folder).file("Archives/John4D.prefs").create()
```

In addition, file and folder objects support `fileSystems`, which provide contextual path to main application folders.

## Filesystem pathnames

4D accepts several `filesystem` pathnames that designate specific 4D folders with variable location on macOS and Windows. Filesystem pathnames are useful for two main reasons:

- Independence: You can move your solution from one place to another regardless of the OS, without having to worry about paths,
- Security: No code can access elements located above the file system root on the disk (sandboxing).

The following filesystem pathnames are supported:

| <b>filesystem</b> | <b>Designates</b>                                      |
|-------------------|--|
| "/DATA"           | Current data folder                                    |
| "/LOGS"           | Logs folder  |
| "/PACKAGE"        | Project root folder (with or without 4dbase extension) |
| "/PROJECT"        | Project folder   |
| "/RESOURCES"      | Current project resources folder                       |
| "/SOURCES"        | Current project sources folder                         |

## POSIX syntax

The POSIX syntax is supported on all platforms. **POSIX syntax is recommended** since it is the most flexible. It is used by default (returned by [file.path](#) and [folder.path](#) properties).

With this syntax:

- folders are separated by "/"
- absolute pathnames start with a "/"
- to move up one folder in a relative path, use "../" in front of the pathname (for security, you cannot move up the filesystem).

In POSIX syntax, you will generally use `filesystem` pathnames with [File](#) and [Folder](#) commands, for example:

```
$pathFile:=File("/DATA/Archives/file 2.txt")
$pathFolder:=Folder("/RESOURCES/Pictures")
```

## Platform-specific syntax

Platform-specific syntax depends on the operating system on which the command is

executed. Note that when creating a file or folder object with this syntax, you must declare it using the `fk platform path` constant as parameter.

## Windows

The following patterns are supported:

- folder separators are ":"
- the text contains ":" and "¥" as the second and third character,
- the text starts with "¥".

Examples with `Folder`:

```
$ok:=Folder("C:\\Monday";fk platform path).create()  
$ok:=Folder("\\\\svr-internal\\tempo";fk platform path).create()
```

## Windows pathnames and escape sequences

The 4D language allows the use of [escape sequences](#). Escape sequences begin with a backslash \, followed by a character. For example, \t is the escape sequence for the Tab character.

Since the \ character is also used as the separator in pathnames in Windows, you need to enter a double \\ in windows pathnames.

## macOS

The following patterns are supported (HFS+ syntax):

- folder separators are ":"
- the path must not start with a ":"

Examples with `Folder`:

```
$ok:=Folder("macintosh hd:";fk platform path).create()  
$ok:=Folder("Monday:Tuesday";fk platform path).create() //a volume must  
be called Monday
```

## Absolute and relative pathnames

### `File` and `Folder` constructors

`File` and `Folder` commands only accept **absolute pathnames**. Relative pathnames are not supported and will return errors. For example, the following code is not allowed:

```
//ERROR  
$ko:=Folder("myFolder").create() //relative pathname with constructor
```

If you want to handle files or folders in various locations (project folder, system folders, etc.), you can use `filesystems` (see above). For example, you can write:

```
$okFolder:=Folder("/PACKAGE/myFolder").create() //folder created at the  
structure level  
$okFile:=File("/DATA/Prefs/tempo.txt").create() //file created in the  
data folder
```

### `.file()` and `.folder()` folder functions

Functions of folder objects such as `folder.file()` and `folder.folder()` expect relative POSIX pathnames. For example:

```
//to reference a "Picture" folder within the user documents folder  
$userImages:=Folder(fk documents folder).folder("Pictures")  
    //to create a folder on the desktop  
$ok:=Folder(fk desktop folder).folder("myFolder").create()
```

Absolute pathnames are not supported and will return errors.

## Examples

The flexibility of file and folder functions offers you various possibilities for handling files and folders, like in the following examples:

```
$f:=Folder(fk desktop folder).folder("archive/jan2019")  
  
$f2:=Folder("/DATA/archive/jan2019").file("total.txt")  
  
$f3:=Folder("/DATA/archive/jan2019")  
  
$f4:=File("/DATA/info.txt")  
  
$f5:=File("c:\\archives\\local\\jan2019.txt";fk platform path)  
  
$f6:=File(fk backup log file)
```

## ORDA

ORDA stands for **Object Relational Data Access**. It is an enhanced technology allowing to access both the model and the data of a database through objects.

Relations are transparently included in the concept, in combination with [lazy loading](#), to remove all the typical hassles of data selection or transfer from the developer.

With ORDA, data is accessed through an abstraction layer, the [datastore](#). A datastore is an object that provides an interface to the database model and data through objects and classes. For example, a table is mapped to a [dataclass](#) object, a field is an [attribute](#) of a dataclass, and records are accessed through [entities](#) and [entity selections](#).

## Why use ORDA?

Instead of representing information as tables, records, and fields, ORDA uses an alternate approach that more accurately maps data to real-world concepts.

Imagine the ability to denormalize a relational structure, yet not affect efficiency. Imagine describing everything about the business objects in your application in such a way that using the data becomes simple and straightforward and removes the need for a complete understanding of the relational structure.

In the ORDA data model, a single dataclass can incorporate all of the elements that make up a traditional relational database table, but can also include values from related parent entities, and direct references to related entities and entity selections.

A query returns a list of entities called an entity selection, which fulfills the role of a SQL query's row set. The difference is that each entity "knows" where it belongs in the data model and "understands" its relationship to all other entities. This means that a developer does not need to explain in a query how to relate the various pieces of information, nor in an update how to write modified values back to the relational structure.

In addition, ORDA objects such as entity selections or entities can be easily bound to UI objects such as list boxes or variables. Combined with powerful features such as the [This](#) and [Form](#) commands, they allow the building modern and modular interfaces based upon objects and collections.

## How to use ORDA?

Basically, ORDA handles objects. In ORDA, all main concepts, including the datastore itself, are available through objects. In 4D, the datastore is automatically [mapped upon the 4D structure](#).

ORDA objects can be handled like 4D standard objects, but they automatically benefit from specific properties and methods.

ORDA objects are created and instantiated when necessary by 4D methods (you do not need to create them). However, ORDA data model objects are associated with [classes where you can add custom functions](#).

## Data Model Objects

The ORDA technology is based upon an automatic mapping of an underlying database structure. It also provides access to data through entity and entity selection objects. As a result, ORDA exposes the whole database as a set of data model objects.

## Structure mapping

When you call a datastore using the `ds` or the `Open datastore` command, 4D automatically references tables and fields of the corresponding 4D structure as properties of the returned `datastore` object:

- Tables are mapped to dataclasses.
- Fields are mapped to storage attributes.
- Relations are mapped to relation attributes - relation names, defined in the Structure editor, are used as relation attribute names.

## General rules

The following rules are applied for any conversions:

- Table, field, and relation names are mapped to object property names. Make sure that such names comply with general object naming rules, as explained in the [object naming conventions](#) section.
- A datastore only references tables with a single primary key. The following tables are not referenced:
  - Tables without a primary key
  - Tables with composite primary keys.
- BLOB fields are automatically available as attributes of the [Blob object](#) type.

ORDA mapping does not take into account:

- the "Invisible" option for tables or fields,
- the virtual structure defined through `SET TABLE TITLES` or `SET FIELD TITLES`,
- the "Manual" or "Automatic" property of relations.

## Rules for remote access control

When accessing a remote datastore through the `Open datastore` command or [REST requests](#), only tables and fields with the **Expose as REST resource** property are available remotely.

This option must be selected at the 4D structure level for each table and each field that you want to be exposed as dataclass and attribute in the datastore:

## Data model update

Any modifications applied at the level of the database structure invalidate the current ORDA model layer. These modifications include:

- adding or removing a table, a field, or a relation

- renaming of a table, a field, or a relation
- changing a core property of a field (type, unique, index, autoincrement, null value support)

When the current ORDA model layer has been invalidated, it is automatically reloaded and updated in subsequent calls of the local `ds` datastore on 4D and 4D Server. Note that existing references to ORDA objects such as entities or entity selections will continue to use the model from which they have been created, until they are regenerated.

However, the updated ORDA model layer is not automatically available in the following contexts:

- a remote 4D application connected to 4D Server -- the remote application must reconnect to the server.
- a remote datastore opened using `Open datastore` or through [REST calls](#) -- a new session must be opened.

## Object definition

### Datastore

The datastore is the interface object to a database. It builds a representation of the whole database as object. A datastore is made of a **model** and **data**:

- The model contains and describes all the dataclasses that make up the datastore. It is independant from the underlying database itself.
- Data refers to the information that is going to be used and stored in this model. For example, names, addresses, and birthdates of employees are pieces of data that you can work with in a datastore.

When handled through the code, the datastore is an object whose properties are all of the [dataclasses](#) which have been specifically exposed.

4D allows you to handle the following datastores:

- the local datastore, based on the current 4D database, returned by the `ds` command (the main datastore).
- one or more remote datastore(s) exposed as REST resources in remote 4D databases, returned by the `Open datastore` command.

A datastore references only a single local or remote database.

The datastore object itself cannot be copied as an object:

```
$mydatastore:=OB Copy(ds) //returns null
```

The datastore properties are however enumerable:

```
ARRAY TEXT($prop;0)
OB GET PROPERTY NAMES(ds;$prop)
//$prop contains the names of all the dataclasses
```

The main (default) datastore is always available through the `ds` command, but the `Open datastore` command allows referencing any remote datastore.

### Dataclass

A dataclass is the equivalent of a table. It is used as an object model and references all fields as attributes, including relational attributes (attributes built upon relations

between dataclasses). Relational attributes can be used in queries like any other attribute.

All dataclasses in a 4D project are available as a property of the `ds` datastore. For remote datastores accessed through [Open datastore](#) or [REST requests](#), the **Expose as REST resource** option must be selected at the 4D structure level for each exposed table that you want to be exposed as dataclass in the datastore.

For example, consider the following table in the 4D structure:

The `Company` table is automatically available as a dataclass in the `ds` datastore. You can write:

```
var $compClass : cs.Company //declares a $compClass object variable of  
the Company class  
$compClass:=ds.Company //assigns the Company dataclass reference to  
$compClass
```

A dataclass object can contain:

- attributes
- relation attributes

The dataclass offers an abstraction of the physical database and allows handling a conceptual data model. The dataclass is the only means to query the datastore. A query is done from a single dataclass. Queries are built around attributes and relation attribute names of the dataclasses. So the relation attributes are the means to involve several linked tables in a query.

The dataclass object itself cannot be copied as an object:

```
$mydataclass:=OB Copy(ds.Employee) //returns null
```

The dataclass properties are however enumerable:

```
ARRAY TEXT($prop;0)  
OB GET PROPERTY NAMES(ds.Employee;$prop)  
//$prop contains the names of all the dataclass attributes
```

## Attribute

Dataclass properties are attribute objects describing the underlying fields or relations. For example:

```
$nameAttribute:=ds.Company.name //reference to class attribute  
$revenuesAttribute:=ds.Company["revenues"] //alternate way
```

This code assigns to `$nameAttribute` and `$revenuesAttribute` references to the name and revenues attributes of the `Company` class. This syntax does NOT return values held inside of the attribute, but instead returns references to the attributes themselves. To handle values, you need to go through [Entities](#).

All eligible fields in a table are available as attributes of their parent [dataclass](#). For remote datastores accessed through [Open datastore](#) or [REST requests](#), the **Expose as REST resource** option must be selected at the 4D structure level for each field that you want to be exposed as a dataclass attribute.

## Storage and Relation attributes

Dataclass attributes come in several kinds: storage, relatedEntity, and relatedEntities. Attributes that are scalar (*i.e.*, provide only a single value) support all the standard 4D data types (integer, text, object, etc.).

- A **storage attribute** is equivalent to a field in the 4D database and can be indexed. Values assigned to a storage attribute are stored as part of the entity when it is saved. When a storage attribute is accessed, its value comes directly from the datastore. Storage attributes are the most basic building block of an entity and are defined by name and data type.
- A **relation attribute** provides access to other entities. Relation attributes can result in either a single entity (or no entity) or an entity selection (0 to N entities). Relation attributes are built upon "classic" relations in the relational structure to provide direct access to related entity or related entities. Relation attributes are directly available in ORDA using their names.

For example, consider the following partial database structure and the relation properties:

All storage attributes will be automatically available:

- in the Project dataclass: "ID", "name", and "companyID"
- in the Company dataclass: "ID", "name", and "discount"

In addition, the following relation attributes will also be automatically available:

- in the Project dataclass: **theClient** attribute, of the "relatedEntity" kind; there is at most one Company for each Project (the client)
- in the Company dataclass: **companyProjects** attribute, of the "relatedEntities" kind; for each Company there is any number of related Projects.

The Manual or Automatic property of a database relation has no effect in ORDA.

All dataclass attributes are exposed as properties of the dataclass:

Keep in mind that these objects describe attributes, but do not give access to data. Reading or writing data is done through [entity objects](#).

## Computed attributes

[Computed attributes](#) are declared using a `get <attributeName>` function in the [Entity class definition](#). Their value is not stored but evaluated each time they are accessed. They do not belong to the underlying database structure, but are built upon it and can be used as any attribute of the data model.

## Entity

An entity is the equivalent of a record. It is actually an object that references a record in the database. It can be seen as an instance of a [dataclass](#), like a record of the table matching the dataclass. However, an entity also contains data correlated to the database related to the datastore.

The purpose of the entity is to manage data (create, update, delete). When an entity reference is obtained by means of an entity selection, it also retains information about the entity selection which allows iteration through the selection.

The entity object itself cannot be copied as an object:

```
$myentity:=OB Copy(ds.Employee.get(1)) //returns null
```

The entity properties are however enumerable:

```
ARRAY TEXT($prop;0)
OB GET PROPERTY NAMES(ds.Employee.get(1);$prop)
//$prop contains the names of all the entity attributes
```

## Entity selection

An entity selection is an object containing one or more reference(s) to entities belonging to the same dataclass. It is usually created as a result of a query or returned from a relation attribute. An entity selection can contain 0, 1 or X entities from the dataclass -- where X can represent the total number of entities contained in the dataclass.

Example:

```
var $e : cs.EmployeeSelection //declares a $e object variable of the EmployeeSelection class type
$e:=ds.Employee.all() //assigns the resulting entity selection reference to the $e variable
```

Entity selections can be "sorted" or "unsorted" ([see below](#)).

Entity selections can also be "shareable" or "non-shareable", depending on [how they have been created](#).

The entity selection object itself cannot be copied as an object:

```
$myentitysel:=OB Copy(ds.Employee.all()) //returns null
```

The entity selection properties are however enumerable:

```
ARRAY TEXT($prop;0)
OB GET PROPERTY NAMES(ds.Employee.all();$prop)
//$prop contains the names of the entity selection properties
//("length", "00", "01"...)
```

## Ordered or unordered entity selection

For optimization reasons, by default 4D ORDA usually creates unordered entity selections, except when you use the `orderBy()` method or use specific options. In this documentation, unless specified, "entity selection" usually refers to an "unordered entity selection".

Ordered entity selections are created only when necessary or when specifically requested using options, i.e. in the following cases:

- result of an `orderBy()` on a selection (of any type) or an `orderBy()` on a dataclass
- result of the `newSelection()` method with the `dk keep ordered` option

Unordered entity selections are created in the following cases:

- result of a standard `query()` on a selection (of any type) or a `query()` on a dataclass,
- result of the `newSelection()` method without option,
- result of any of the comparison methods, whatever the input selection types:

`or()`, `and()`, `minus()`.

The following entity selections are always **ordered**:

- entity selections returned by 4D Server to a remote client
- entity selections built upon remote datastores.

Note that when an ordered entity selection becomes an unordered entity selection, any repeated entity references are removed.

## Data Model Classes

ORDA allows you to create high-level class functions above the data model. This allows you to write business-oriented code and "publish" it just like an API. Datastore, dataclasses, entity selections, and entities are all available as class objects that can contain functions.

For example, you could create a `getNextWithHigherSalary()` function in the `EmployeeEntity` class to return employees with a salary higher than the selected one. It would be as simple as calling:

```
$nextHigh:=ds.Employee.get(1).getNextWithHigherSalary()
```

Developers can not only use these functions in local datastores, but also in client/server and remote architectures:

```
//$cityManager is the reference of a remote datastore  
Form.comp.city:=$cityManager.City.getCityName(Form.comp.zipcode)
```

Thanks to this feature, the entire business logic of your 4D application can be stored as a independent layer so that it can be easily maintained and reused with a high level of security:

- You can "hide" the overall complexity of the underlying physical structure and only expose understandable and ready-to-use functions.
- If the physical structure evolves, you can simply adapt function code and client applications will continue to call them transparently.
- By default, all of your data model class functions (including [computed attribute functions](#)) and [alias attributes](#) are **not exposed** to remote applications and cannot be called from REST requests. You must explicitly declare each public function and alias with the [exposed](#) keyword.

In addition, 4D [automatically pre-creates](#) the classes for each available data model object.

## Architecture

ORDA provides **generic classes** exposed through the `4D class store`, as well as **user classes** (extending generic classes) exposed in the `cs class store`:

All ORDA data model classes are exposed as properties of the `cs` class store. The following ORDA classes are available:

| Class                     | Example name         | Instantiated by   |
|---------------------------|----------------------|---|
| cs.DataStore              | cs.DataStore         | <code>ds</code> command<br><code>dataStore.DataClassName</code> ,<br><code>dataStore["DataClassName"]</code>  |
| cs.DataClassName          | cs.Employee          | <code>dataClass.get()</code> , <code>dataClass.new()</code> ,<br><code>entitySelection.first()</code> ,<br><code>entitySelection.last()</code> ,<br><code>entity.previous()</code> , <code>entity.next()</code> ,<br><code>entity.first()</code> , <code>entity.last()</code> ,<br><code>entity.clone()</code>  |
| cs.DataClassNameEntity    | cs.EmployeeEntity    | <code>dataClass.query()</code> ,<br><code>entitySelection.query()</code> ,<br><code>dataClass.all()</code> ,<br><code>dataClass.fromCollection()</code> ,<br><code>dataClass.newSelection()</code> ,<br><code>entitySelection.drop()</code> ,<br><code>entity.getSelection()</code> ,<br><code>entitySelection.and()</code> ,<br><code>entitySelection.minus()</code> ,<br><code>entitySelection.or()</code> ,<br><code>entitySelection.orderBy()</code> ,<br><code>entitySelection.orderByFormula()</code> ,<br><code>entitySelection.slice()</code> , Create entity selection |
| cs.DataClassNameSelection | cs.EmployeeSelection |   |

ORDA user classes are stored as regular class files (.4dm) in the Classes subfolder of the project ([see below](#)).

Also, object instances from ORDA data model user classes benefit from their parent's properties and functions:

- a Datastore class object can call functions from the [ORDA Datastore generic class](#).
- a Dataclass class object can call functions from the [ORDA Dataclass generic class](#).
- an Entity selection class object can call functions from the [ORDA Entity selection generic class](#).
- an Entity class object can call functions from the [ORDA Entity generic class](#).

## Class Description

- History

### DataStore Class

A 4D database exposes its own DataStore class in the `cs` class store.

- **Extends:** 4D.DataStoreImplementation
- **Class name:** cs.DataStore

You can create functions in the DataStore class that will be available through the `ds` object.

### Example

```
// cs.DataStore class

Class extends DataStoreImplementation

Function getDesc
$0:="Database exposing employees and their companies"
```

This function can then be called:

```
$desc:=ds.getDesc() // "Database exposing..."
```

## DataClass Class

Each table exposed with ORDA offers a DataClass class in the `cs` class store.

- **Extends:** 4D.DataClass
- **Class name:** `cs.DataClassName` (where `DataClassName` is the table name)
- **Example name:** `cs.Employee`

### Example

```
// cs.Company class

Class extends DataClass

// Returns companies whose revenue is over the average
// Returns an entity selection related to the Company DataClass

Function GetBestOnes()
$sel:=This.query("revenues >= :1";This.all().average("revenues"));
$0:=$sel
```

Then you can get an entity selection of the "best" companies by executing:

```
var $best : cs.CompanySelection
$best:=ds.Company.GetBestOnes()
```

[Computed attributes](#) are defined in the [Entity Class](#).

### Example with a remote datastore

The following `City` catalog is exposed in a remote datastore (partial view):

The `City Class` provides an API:

```
// cs.City class

Class extends DataClass

Function getCityName()
    var $1; $zipcode : Integer
    var $zip : 4D.Entity
    var $0 : Text

    $zipcode:=$1
    $zip:=ds.ZipCode.get($zipcode)
    $0:=""

    If ($zip#Null)
        $0:=$zip.city.name
    End if
```

The client application opens a session on the remote datastore:

```
$cityManager:=Open datastore(New
object("hostname";"127.0.0.1:8111");"CityManager")
```

Then a client application can use the API to get the city matching a zip code (for example) from a form:

```
Form.comp.city:=$cityManager.City.getCityName(Form.comp.zipcode)
```

## EntitySelection Class

Each table exposed with ORDA offers an EntitySelection class in the `cs` class store.

- **Extends:** 4D.EntitySelection
- **Class name:** *DataClassNameSelection* (where *DataClassName* is the table name)
- **Example name:** cs.EmployeeSelection

### Example

```
// cs.EmployeeSelection class

Class extends EntitySelection

//Extract the employees with a salary greater than the average from
this entity selection

Function withSalaryGreaterThanAverage
    C_OBJECT($0)
    $0:=This.query("salary >
:1";This.average("salary")).orderBy("salary")
```

Then you can get employees with a salary greater than the average in any entity selection by executing:

```
$moreThanAvg:=ds.Company.all().employees.withSalaryGreaterThanAverage()
```

## Entity Class

Each table exposed with ORDA offers an Entity class in the `cs` class store.

- **Extends:** 4D.Entity
- **Class name:** *DataClassNameEntity* (where *DataClassName* is the table name)

- **Example name:** cs.CityEntity

## Computed attributes

Entity classes allow you to define **computed attributes** using specific keywords:

- Function get *attributeName*
- Function set *attributeName*
- Function query *attributeName*
- Function orderBy *attributeName*

For information, please refer to the [Computed attributes](#) section.

## Alias attributes

Entity classes allow you to define **alias attributes**, usually over related attributes, using the `Alias` keyword:

`Alias attributeName targetPath`

For information, please refer to the [Alias attributes](#) section.

## Example

```
// cs.CityEntity class

Class extends Entity

Function getPopulation()
    $0:=This.zips.sum("population")

Function isBigCity(): Boolean
// The getPopulation() function is usable inside the class
$0:=This.getPopulation()>50000
```

Then you can call this code:

```
var $cityManager; $city : Object

$cityManager:=Open datastore(New
object("hostname";"127.0.0.1:8111");"CityManager")
$city:=$cityManager.City.getCity("Caguas")

If ($city.isBigCity())
    ALERT($city.name + " is a big city")
End if
```

## Specific rules

When creating or editing data model classes, you must pay attention to the following rules:

- Since they are used to define automatic DataClass class names in the [cs class store](#), 4D tables must be named in order to avoid any conflict in the `cs` namespace. In particular:
  - Do not give the same name to a 4D table and to a [user class name](#). If such a case occurs, the constructor of the user class becomes unusable (a warning is returned by the compiler).
  - Do not use a reserved name for a 4D table (e.g., "DataClass").

- When defining a class, make sure the `Class extends` statement exactly matches the parent class name (remember that they're case sensitive). For example, `Class extends EntitySelection` for an entity selection class.
- You cannot instantiate a data model class object with the `new()` keyword (an error is returned). You must use a regular method as listed in the [Instantiated by column of the ORDA class table](#).
- You cannot override a native ORDA class function from the [4D class store](#) with a data model user class function.

## Preemptive execution

When compiled, data model class functions are executed:

- in **preemptive or cooperative processes** (depending on the calling process) in single-user applications,
- in **preemptive processes** in client/server applications (except if the `local` keyword is used, in which case it depends on the calling process like in single-user).

If your project is designed to run in client/server, make sure your data model class function code is thread-safe. If thread-unsafe code is called, an error will be thrown at runtime (no error will be thrown at compilation time since cooperative execution is supported in single-user applications).

## Computed attributes

### Overview

A computed attribute is a dataclass attribute with a data type that masks a calculation. [Standard 4D classes](#) implement the concept of computed properties with `get` (*getter*) and `set` (*setter*) [accessor functions](#). ORDA dataclass attributes benefit from this feature and extend it with two additional functions: `query` and `orderBy`.

At the very minimum, a computed attribute requires a `get` function that describes how its value will be calculated. When a *getter* function is supplied for an attribute, 4D does not create the underlying storage space in the datastore but instead substitutes the function's code each time the attribute is accessed. If the attribute is not accessed, the code never executes.

A computed attribute can also implement a `set` function, which executes whenever a value is assigned to the attribute. The *setter* function describes what to do with the assigned value, usually redirecting it to one or more storage attributes or in some cases other entities.

Just like storage attributes, computed attributes may be included in **queries**. By default, when a computed attribute is used in a ORDA query, the attribute is calculated once per entity examined. In some cases this is sufficient. However for better performance, especially in client/server, computed attributes can implement a `query` function that relies on actual dataclass attributes and benefits from their indexes.

Similarly, computed attributes can be included in **sorts**. When a computed attribute is used in a ORDA sort, the attribute is calculated once per entity examined. Just like in queries, computed attributes can implement an `orderBy` function that substitutes other attributes during the sort, thus increasing performance.

## How to define computed attributes

You create a computed attribute by defining a `get` accessor in the [entity class](#) of the dataclass. The computed attribute will be automatically available in the dataclass attributes and in the entity attributes.

Other computed attribute functions (`set`, `query`, and `orderBy`) can also be defined in the entity class. They are optional.

Within computed attribute functions, [`This`](#) designates the entity. Computed attributes can be used and handled as any dataclass attribute, i.e. they will be processed by [entity class](#) or [entity selection class](#) functions.

ORDA computed attributes are not [exposed](#) by default. You expose a computed attribute by adding the `exposed` keyword to the [get function](#) definition.

**get and set functions** can have the [local](#) property to optimize client/server processing.

```
Function get <attributeName>
```

### Syntax

```
{local} {exposed} Function get <attributeName>({$event : Object}) ->
$result : type
// code
```

The *getter* function is mandatory to declare the *attributeName* computed attribute. Whenever the *attributeName* is accessed, 4D evaluates the `Function get` code and returns the *\$result* value.

A computed attribute can use the value of other computed attribute(s). Recursive calls generate errors.

The *getter* function defines the data type of the computed attribute thanks to the *\$result* parameter. The following resulting types are allowed:

- Scalar (text, boolean, date, time, number)
- Object
- Image
- BLOB
- Entity (i.e. cs.EmployeeEntity)
- Entity selection (i.e. cs.EmployeeSelection)

The *\$event* parameter contains the following properties:

| Property      | Type    | Description   |
|---------------|---------|---|
| attributeName | Text    | Computed attribute name   |
| dataClassName | Text    | Dataclass name  |
| kind          | Text    | "get"   |
| result        | Variant | Optional. Add this property with Null value if you want a scalar attribute to return Null |

### Examples

- *fullName* computed attribute:

```

Function get fullName($event : Object) -> $fullName : Text

Case of
  : (This.firstName=NULL) & (This.lastName=NULL)
    $event.result:=NULL //use result to return NULL
  : (This.firstName=NULL)
    $fullName:=This.lastName
  : (This.lastName=NULL)
    $fullName:=This.firstName
Else
  $fullName:=This.firstName+ " "+This.lastName
End case

```

- A computed attribute can be based upon an entity related attribute:

```

Function get bigBoss($event : Object) -> $result: cs.EmployeeEntity
  $result:=This.manager.manager

```

- A computed attribute can be based upon an entity selection related attribute:

```

Function get coWorkers($event : Object) -> $result: cs.EmployeeSelection
  If (This.manager.manager=NULL)
    $result:=ds.Employee.newSelection()
  Else
    $result:=This.manager.directReports.minus(this)
  End if

```

### **Function set <attributeName>**

#### **Syntax**

```
{local} Function set <attributeName>($value : type {; $event : Object})
// code
```

The *setter* function executes whenever a value is assigned to the attribute. This function usually processes the input value(s) and the result is dispatched between one or more other attributes.

The *\$value* parameter receives the value assigned to the attribute.

The *\$event* parameter contains the following properties:

| <b>Property</b> | <b>Type</b> | <b>Description</b>                            |
|-----------------|-------------|---|
| attributeName   | Text        | Computed attribute name                       |
| dataClassName   | Text        | Dataclass name                                |
| kind            | Text        | "set"   |
| value           | Variant     | Value to be handled by the computed attribute |

#### **Example**

```

Function set fullName($value : Text; $event : Object)
  var $p : Integer
  $p:=Position(" "; $value)
  This.firstname:=Substring($value; 1; $p-1) // "" if $p<0
  This.lastname:=Substring($value; $p+1)

```

### **Function query <attributeName>**

## Syntax

```
Function query <attributeName>($event : Object)
Function query <attributeName>($event : Object) -> $result : Text
Function query <attributeName>($event : Object) -> $result : Object
// code
```

This function supports three syntaxes:

- With the first syntax, you handle the whole query through the `$event.result` object property.
- With the second and third syntaxes, the function returns a value in `$result`:
  - If `$result` is a Text, it must be a valid query string
  - If `$result` is an Object, it must contain two properties:

| Property                         | Type       | Description   |
|----------------------------------|------------|---|
| <code>\$result.query</code>      | Text       | Valid query string with placeholders (:1, :2, etc.) |
| <code>\$result.parameters</code> | Collection | values for placeholders                             |

The `query` function executes whenever a query using the computed attribute is launched. It is useful to customize and optimize queries by relying on indexed attributes. When the `query` function is not implemented for a computed attribute, the search is always sequential (based upon the evaluation of all values using the `get <AttributeName>` function).

The following features are not supported:

- calling a `query` function on computed attributes of type Entity or Entity selection,
- using the `order by` keyword in the resulting query string.

The `$event` parameter contains the following properties:

| Property                   | Type    | Description   |
|----------------------------|---------|---|
| <code>attributeName</code> | Text    | Computed attribute name   |
| <code>dataClassName</code> | Text    | Dataclass name  |
| <code>kind</code>          | Text    | "query"   |
| <code>value</code>         | Variant | Value to be handled by the computed attribute<br>Query operator (see also the <a href="#">query class function</a> ). Possible values: <ul style="list-style-type: none"><li>== (equal to, @ is wildcard)</li><li>===(equal to, @ is not wildcard)</li><li>!= (not equal to, @ is wildcard)</li><li>!== (not equal to, @ is not wildcard)</li><li>&lt; (less than)</li><li>&lt;= (less than or equal to)</li><li>&gt; (greater than)</li><li>&gt;= (greater than or equal to)</li><li>IN (included in)</li><li>% (contains keyword)</li></ul> |
| <code>operator</code>      | Text    | Value to be handled by the computed attribute. Pass <code>Null</code> in  |
| <code>result</code>        | Variant | this property if you want to let 4D execute the default query (always sequential for computed attributes).  |

If the function returns a value in `$result` and another value is assigned to the

`$event.result` property, the priority is given to `$event.result`.

## Examples

- Query on the *fullName* computed attribute.

```
Function query fullName($event : Object) ->$result : Object

    var $fullname; $firstname; $lastname; $query : Text
    var $operator : Text
    var $p : Integer
    var $parameters : Collection

    $operator:=$event.operator
    $fullname:=$event.value

    $p:=Position(" "; $fullname)
    If ($p>0)
        $firstname:=Substring($fullname; 1; $p-1)+"@"
        $lastname:=Substring($fullname; $p+1)+"@"
        $parameters:=New collection($firstname; $lastname) // two
items collection
    Else
        $fullname:=$fullname+"@"
        $parameters:=New collection($fullname) // single item
collection
    End if

    Case of
    : ($operator=="==") | ($operator=="===")
        If ($p>0)
            $query:="(firstName = :1 and lastName = :2) or (firstName =
:2 and lastName = :1)"
        Else
            $query:="firstName = :1 or lastName = :1"
        End if
    : ($operator!="!=")
        If ($p>0)
            $query:="firstName != :1 and lastName != :2 and firstName
!= :2 and lastName != :1"
        Else
            $query:="firstName != :1 and lastName != :1"
        End if
    End case

    $result:=New object("query"; $query; "parameters"; $parameters)
```

Keep in mind that using placeholders in queries based upon user text input is recommended for security reasons (see [query\(\) description](#)).

Calling code, for example:

```
$emps:=ds.Employee.query("fullName = :1"; "Flora Pionsin")
```

- This function handles queries on the *age* computed attribute and returns an object with parameters:

```

Function query age($event : Object)->$result : Object

    var $operator : Text
    var $age : Integer
    var $_ages : Collection

    $operator:=$event.operator

    $age:=Num($event.value) // integer
    $d1:=Add to date( Current date; -$age-1; 0; 0)
    $d2:=Add to date($d1; 1; 0; 0)
    $parameters:=New collection($d1; $d2)

    Case of

        : ($operator=="==")
            $query:="birthday > :1 and birthday <= :2" // after d1 and
before or egal d2

        : ($operator=="===")
            $query:="birthday = :2" // d2 = second calculated date (=
birthday date)

        : ($operator==">=")
            $query:="birthday <= :2"

        //... other operators

    End case

    If (Undefined($event.result))
        $result:=New object
        $result.query:=$query
        $result.parameters:=$parameters
    End if

```

### Calling code, for example:

```

// people aged between 20 and 21 years (-1 day)
$twenty:=people.query("age = 20") // calls the "==" case

// people aged 20 years today
$twentyToday:=people.query("age === 20") // equivalent to
people.query("age is 20")

```

## Function `orderBy <attributeName>`

### Syntax

```

Function orderBy <attributeName>($event : Object)
Function orderBy <attributeName>($event : Object)-> $result : Text

// code

```

The `orderBy` function executes whenever the computed attribute needs to be ordered. It allows sorting the computed attribute. For example, you can sort *fullName* on first names then last names, or conversely. When the `orderBy` function is not implemented for a computed attribute, the sort is always sequential (based upon the

evaluation of all values using the `get <AttributeName>` function).

Calling an `orderBy` function on computed attributes of type Entity class or Entity selection class **is not supported**.

The `$event` parameter contains the following properties:

| Property      | Type    | Description   |
|---------------|---------|---|
| attributeName | Text    | Computed attribute name   |
| dataClassName | Text    | Dataclass name  |
| kind          | Text    | "orderBy"   |
| value         | Variant | Value to be handled by the computed attribute   |
| operator      | Text    | "desc" or "asc" (default)   |
| descending    | Boolean | <code>true</code> for descending order, <code>false</code> for ascending order  |
| result        | Variant | Value to be handled by the computed attribute. Pass <code>Null</code> if you want to let 4D execute the default sort. |

You can use either the `operator` or the `descending` property. It is essentially a matter of programming style (see examples).

You can return the `orderBy` string either in the `$event.result` object property or in the `$result` function result. If the function returns a value in `$result` and another value is assigned to the `$event.result` property, the priority is given to `$event.result`.

## Example

You can write conditional code:

```
Function orderBy fullName($event : Object)-> $result : Text
    If ($event.descending=True)
        $result:="firstName desc, lastName desc"
    Else
        $result:="firstName, lastName"
    End if
```

You can also write compact code:

```
Function orderBy fullName($event : Object)-> $result : Text
    $result:="firstName "+$event.operator+", "lastName
"+$event.operator
```

Conditional code is necessary in some cases:

```
Function orderBy age($event : Object)-> $result : Text
    If ($event.descending=True)
        $result:="birthday asc"
    Else
        $result:="birthday desc"
    End if
```

## Alias attributes

### Overview

An **alias** attribute is built above another attribute of the data model, named **target** attribute. The target attribute can belong to a related dataclass (available through any number of relation levels) or to the same dataclass. An alias attribute stores no data,

but the path to its target attribute. You can define as many alias attributes as you want in a dataclass.

Alias attributes are particularly useful to handle N to N relations. They bring more readability and simplicity in the code and in queries by allowing to rely on business concepts instead of implementation details.

## How to define alias attributes

You create an alias attribute in a dataclass by using the `Alias` keyword in the [entity class](#) of the dataclass.

```
Alias <attributeName> <targetPath>
```

### Syntax

```
{exposed} Alias <attributeName> <targetPath>
```

*attributeName* must comply with [standard rules for property names](#).

*targetPath* is an attribute path containing one or more levels, such as "employee.company.name". If the target attribute belongs to the same dataclass, *targetPath* is the attribute name.

An alias can be used as a part of a path of another alias.

A [computed attribute](#) can be used in an alias path, but only as the last level of the path, otherwise, an error is returned. For example, if "fullName" is a computed attribute, an alias with path "employee.fullName" is valid.

ORDA alias attributes are **not exposed** by default. You must add the `exposed` keyword before the `Alias` keyword if you want the alias to be available to remote requests.

## Using alias attributes

Alias attributes are read-only (except when based upon a scalar attribute of the same dataclass, see the last example below). They can be used instead of their target attribute path in class functions such as:

### Function

```
dataClass.query(), entitySelection.query()  
entity.toObject()  
entitySelection.toCollection()  
entitySelection.extract()  
entitySelection.orderBy()  
entitySelection.orderByFormula()  
entitySelection.average()  
entitySelection.count()  
entitySelection.distinct()  
entitySelection.sum()  
entitySelection.min()  
entitySelection.max()  
entity.diff()  
entity.touchedAttributes()
```

Keep in mind that alias attributes are calculated on the server. In remote configurations, updating alias attributes in entities requires that entities are reloaded from the server.

## Alias properties

Alias attribute `kind` is "alias".

An alias attribute inherits its data `type` property from the target attribute:

- if the target attribute `kind` is "storage", the alias data type is of the same type,
- if the target attribute `kind` is "relatedEntity" or "relatedEntities", the alias data type is of the `4D.Entity` or `4D.EntitySelection` type ("classnameEntity" or "classnameSelection").

Alias attributes based upon relations have a specific `path` property, containing the path of their target attributes. Alias attributes based upon attributes of the same dataclass have the same properties as their target attributes (and no `path` property).

## Examples

Considering the following model:

In the Teacher dataclass, an alias attribute returns all students of a teacher:

```
// cs.TeacherEntity class  
  
Class extends Entity  
  
Alias students courses.student //relatedEntities
```

In the Student dataclass, an alias attribute returns all teachers of a student:

```
// cs.StudentEntity class  
  
Class extends Entity  
  
Alias teachers courses.teacher //relatedEntities
```

In the Course dataclass:

- an alias attribute returns another label for the "name" attribute
- an alias attribute returns the teacher name
- an alias attribute returns the student name

```
// cs.CourseEntity class  
  
Class extends Entity  
  
Exposed Alias courseName name //scalar  
Exposed Alias teacherName teacher.name //scalar value  
Exposed Alias studentName student.name //scalar value
```

You can then execute the following queries:

```

// Find course named "Archaeology"
ds.Course.query("courseName = :1";"Archaeology")

// Find courses given by the professor Smith
ds.Course.query("teacherName = :1";"Smith")

// Find courses where Student "Martin" assists
ds.Course.query("studentName = :1";"Martin")

// Find students who have M. Smith as teacher
ds.Student.query("teachers.name = :1";"Smith")

// Find teachers who have M. Martin as Student
ds.Teacher.query("students.name = :1";"Martin")
// Note that this very simple query string processes a complex
// query including a double join, as you can see in the queryPlan:
// "Join on Table : Course : Teacher.ID = Course.teacherID,
// subquery:[ Join on Table : Student : Course.studentID =
Student.ID,
// subquery:[ Student.name === Martin]]"

```

You can also edit the value of the `courseName` alias:

```

// Rename a course using its alias attribute
$arch:=ds.Course.query("courseName = :1";"Archaeology")
$arch.courseName:="Archaeology II"
$arch.save() //courseName and name are "Archaeology II"

```

## Exposed vs non-exposed functions

For security reasons, all of your data model class functions and alias attributes are **not exposed** (i.e., private) by default to remote requests.

Remote requests include:

- Requests sent by remote 4D applications connected through [Open datastore](#)
- REST requests

Regular 4D client/server requests are not impacted. Data model class functions are always available in this architecture.

A function that is not exposed is not available on remote applications and cannot be called on any object instance from a REST request. If a remote application tries to access a non-exposed function, the "-10729 - Unknown member method" error is returned.

To allow a data model class function to be called by a remote request, you must explicitly declare it using the `exposed` keyword. The formal syntax is:

```
// declare an exposed function
exposed Function <functionName>
```

The `exposed` keyword can only be used with Data model class functions. If used with a [regular user class](#) function, it is ignored and an error is returned by the compiler.

## Example

You want an exposed function to use a private function in a dataclass class:

```

Class extends DataClass

//Public function
exposed Function registerNewStudent($student : Object) -> $status : Object

var $entity : cs.StudentsEntity

$entity:=ds.Students.new()
$entity.fromObject($student)
$entity.school:=This.query("name=:1"; $student.schoolName).first()
$entity.serialNumber:=This.computeSerialNumber()
$status:=$entity.save()

//Not exposed (private) function
Function computeIDNumber()-> $id : Integer
//compute a new ID number
$id:=...

```

When the code is called:

```

var $remoteDS; $student; $status : Object
var $id : Integer

$remoteDS:=Open datastore(New object("hostname"; "127.0.0.1:8044");
"students")
$student:=New object("firstname"; "Mary"; "lastname"; "Smith";
"schoolName"; "Math school")

$status:=$remoteDS.Schools.registerNewStudent($student) // OK
$id:=$remoteDS.Schools.computeIDNumber() // Error "Unknown member
method"

```

## Local functions

By default in client/server architecture, ORDA data model functions are executed **on the server**. It usually provides the best performance since only the function request and the result are sent over the network.

However, it could happen that a function is fully executable on the client side (e.g., when it processes data that's already in the local cache). In this case, you can save requests to the server and thus, enhance the application performance by inserting the `local` keyword. The formal syntax is:

```
// declare a function to execute locally in client/server
local Function <functionName>
```

With this keyword, the function will always be executed on the client side.

The `local` keyword can only be used with data model class functions. If used with a [regular user class](#) function, it is ignored and an error is returned by the compiler.

Note that the function will work even if it eventually requires to access the server (for example if the ORDA cache is expired). However, it is highly recommended to make sure that the local function does not access data on the server, otherwise the local execution could not bring any performance benefit. A local function that generates many requests to the server is less efficient than a function executed on the server that would only return the resulting values. For example, consider the following function on the Schools entity class:

```
// Get the youngest students
// Inappropriate use of local keyword
local Function getYoungest
    var $0 : Object
    $0:=This.students.query("birthDate >= :1"; !2000-01-01!).orderBy("birthDate desc").slice(0; 5)
```

- **without** the `local` keyword, the result is given using a single request
- **with** the `local` keyword, 4 requests are necessary: one to get the Schools entity students, one for the `query()`, one for the `orderBy()`, and one for the `slice()`. In this example, using the `local` keyword is inappropriate.

## Examples

### Calculating age

Given an entity with a `birthDate` attribute, we want to define an `age()` function that would be called in a list box. This function can be executed on the client, which avoids triggering a request to the server for each line of the list box.

On the `StudentsEntity` class:

```
Class extends Entity

local Function age() -> $age: Variant

If (This.birthDate#!00-00-00!)
    $age:=Year of(Current date)-Year of(This.birthDate)
Else
    $age:=Null
End if
```

### Checking attributes

We want to check the consistency of the attributes of an entity loaded on the client and updated by the user before requesting the server to save them.

On the `StudentsEntity` class, the local `checkData()` function checks the Student's age:

```
Class extends Entity

local Function checkData() -> $status : Object

$status:=New object("success"; True)
Case of
    : (This.age()=Null)
        $status.success:=False
        $status.statusText:="The birthdate is missing"

    :((This.age() <15) | (This.age()>30) )
        $status.success:=False
        $status.statusText:="The student must be between 15 and 30 -
This one is "+String(This.age())
End case
```

Calling code:

```

var $status : Object

//Form.student is loaded with all its attributes and updated on a Form
$status:=Form.student.checkData()
If ($status.success)
    $status:=Form.student.save() // call the server
End if

```

## Support in 4D IDE

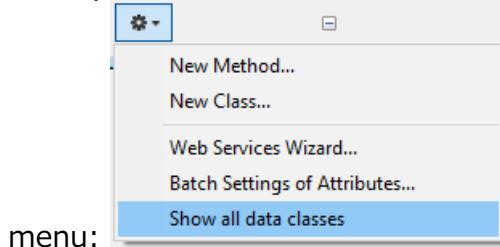
### Class files

An ORDA data model user class is defined by adding, at the [same location as regular class files](#) (i.e. in the `/Sources/Classes` folder of the project folder), a `.4dm` file with the name of the class. For example, an entity class for the `Utilities` dataclass will be defined through a `UtilitiesEntity.4dm` file.

### Creating classes

4D automatically pre-creates empty classes in memory for each available data model object.

By default, empty ORDA classes are not displayed in the Explorer. To show them you need to select **Show all data classes** from the Explorer's options



menu:

ORDA user classes have a different icon from regular classes. Empty classes are dimmed:

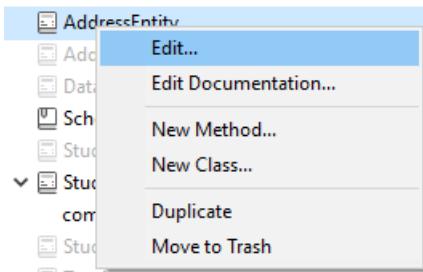
To create an ORDA class file, you just need to double-click on the corresponding predefined class in the Explorer. 4D creates the class file and add the `extends` code. For example, for an Entity class:

```
Class extends Entity
```

Once a class is defined, its name is no longer dimmed in the Explorer.

### Editing classes

To open a defined ORDA class in the 4D Code Editor, select or double-click on an ORDA class name and use **Edit...** from the contextual menu/options menu of the Explorer window:



For ORDA classes based upon the local datastore (`ds`), you can directly access the class code from the 4D Structure window:

## Code Editor

In the 4D Code Editor, variables typed as an ORDA class automatically benefit from autocompletion features. Example with an Entity class variable:

## Working with data

In ORDA, you access data through [entities](#) and [entity selections](#). These objects allow you to create, update, query, or sort the data of the datastore.

### Creating an entity

There are two ways to create a new entity in a dataclass:

- Since entities are references to database records, you can create entities by creating records using the "classic" 4D language and then reference them with ORDA methods such as `entity.next()` or `entitySelection.first()`.
- You can also create an entity using the `dataClass.new()` method.

Keep in mind that the entity is only created in memory. If you want to add it to the datastore, you must call the `entity.save()` method.

Entity attributes are directly available as properties of the entity object. For more information, please refer to [Using entity attributes](#).

For example, we want to create a new entity in the "Employee" dataclass in the current datastore with "John" and "Dupont" assigned to the firstname and name attributes:

```
var $myEntity : cs.EmployeeEntity
$myEntity:=ds.Employee.new() //Create a new object of the entity type
$myEntity.name:="Dupont" //assign 'Dupont' to the 'name' attribute
$myEntity.firstname:="John" //assign 'John' to the 'firstname'
attribute
$myEntity.save() //save the entity
```

An entity is defined only in the process where it was created. You cannot, for example, store a reference to an entity in an interprocess variable and use it in another process.

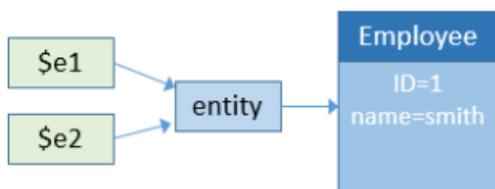
### Entities and references

An entity contains a reference to a 4D record. Different entities can reference the same 4D record. Also, since an entity can be stored in a 4D object variable, different variables can contain a reference to the same entity.

If you execute the following code:

```
var $e1; $e2 : cs.EmployeeEntity
$e1:=ds.Employee.get(1) //access the employee with ID 1
$e2:=$e1
$e1.name:="Hammer"
//both variables $e1 and $e2 share the reference to the same entity
//$e2.name contains "Hammer"
```

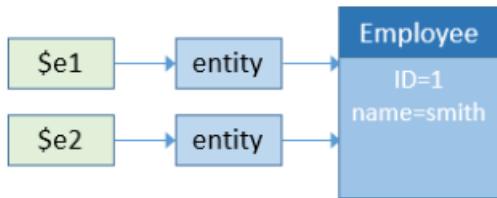
This is illustrated by the following graphic:



Now if you execute:

```
var $e1; $e2 : cs.EmployeeEntity
$e1:=ds.Employee.get(1)
$e2:=ds.Employee.get(1)
$e1.name:="Hammer"
//variable $e1 contains a reference to an entity
//variable $e2 contains another reference to another entity
//$e2.name contains "smith"
```

This is illustrated by the following graphic:



Note however that entities refer to the same record. In all cases, if you call the `entity.save()` method, the record will be updated (except in case of conflict, see [Entity locking](#)).

In fact, `$e1` and `$e2` are not the entity itself, but a reference to the entity. It means that you can pass them directly to any function or method, and it will act like a pointer, and faster than a 4D pointer. For example:

```
For each($entity;$selection)
    do_Capitalize($entity)
End for each
```

And the method is:

```
$entity:=$1
$name:=$entity.lastname
If (Not ($name=NULL))

$name:=Uppercase(Substring($name;1;1))+Lowercase(Substring($name;2))
End if
$entity.lastname:=$name
```

You can handle entities like any other object in 4D and pass their references directly as [parameters](#).

With the entities, there is no concept of "current record" as in the classic 4D language. You can use as many entities as you need, at the same time. There is also no automatic lock on an entity (see [Entity locking](#)). When an entity is loaded, it uses the [lazy loading](#) mechanism, which means that only the needed information is loaded. Nevertheless, in client/server, the entity can be automatically loaded directly if necessary.

## Using entity attributes

Entity attributes store data and map corresponding fields in the corresponding table. Entity attributes of the storage kind can be set or get as simple properties of the entity object, while entity of the **relatedEntity** or **relatedEntities** kind will return an entity or an entity selection.

For more information on the attribute kind, please refer to the [Storage and Relation attributes](#) paragraph.

For example, to set a storage attribute:

```
$entity:=ds.Employee.get(1) //get employee attribute with ID 1
$name:=$entity.lastname //get the employee name, e.g. "Smith"
$entity.lastname:="Jones" //set the employee name
$entity.save() //save the modifications
```

Database Blob fields ([scalar blobs](#)) are automatically converted to and from blob object attributes ([4D.Blob](#)) when handled through ORDA. When saving a blob object attribute, keep in mind that, unlike blob object size which is only limited by the available memory, Blob field size is limited to 2GB.

Accessing a related attribute depends on the attribute kind. For example, with the following structure:

You can access data through the related object(s):

```
$entity:=ds.Project.all().first().theClient //get the Company entity
associated to the project
$EntitySel:=ds.Company.all().first().companyProjects //get the
selection of projects for the company
```

Note that both *theClient* and *companyProjects* in the above example are primary relation attributes and represent a direct relationship between the two dataclasses. However, relation attributes can also be built upon paths through relationships at several levels, including circular references. For example, consider the following structure:

Each employee can be a manager and can have a manager. To get the manager of the manager of an employee, you can simply write:

```
$myEmp:=ds.Employee.get(50)
$manLev2:=$myEmp.manager.manager.lastname
```

## Assigning values to relation attributes

In the ORDA architecture, relation attributes directly contain data related to entities:

- An N->1 type relation attribute (**relatedEntity** kind) contains an entity
- A 1->N type relation attribute (**relatedEntities** kind) contains an entity selection

Let's look at the following (simplified) structure:

In this example, an entity in the "Employee" dataclass contains an object of type Entity in the "employer" attribute (or a null value). An entity in the "Company" dataclass contains an object of type EntitySelection in the "staff" attribute (or a null value).

In ORDA, the Automatic or Manual property of relations has no effect.

To assign a value directly to the "employer" attribute, you must pass an existing entity from the "Company" dataclass. For example:

```

$emp:=ds.Employee.new() // create an employee
$emp.lastname:="Smith" // assign a value to an attribute
$emp.employer:=ds.Company.query("name =:1";"4D")[0] //assign a
company entity
$emp.save()

```

4D provides an additional facility for entering a relation attribute for an N entity related to a "1" entity: you pass the primary key of the "1" entity directly when assigning a value to the relation attribute. For this to work, you pass data of type Number or Text (the primary key value) to the relation attribute. 4D then automatically takes care of searching for the corresponding entity in the dataclass. For example:

```

$emp:=ds.Employee.new()
$emp.lastname:="Wesson"
$emp.employer:=2 // assign a primary key to the relation attribute
    //4D looks for the company whose primary key (in this case, its ID)
is 2
    //and assigns it to the employee
$emp.save()

```

This is particularly useful when you are importing large amounts of data from a relational database. This type of import usually contains an "ID" column, which references a primary key that you can then assign directly to a relation attribute.

This also means that you can assign primary keys in the N entities without corresponding entities having already been created in the 1 datastore class. If you assign a primary key that does not exist in the related datastore class, it is nevertheless stored and assigned by 4D as soon as this "1" entity is created.

You can assign or modify the value of a "1" related entity attribute from the "N" dataclass directly through the related attribute. For example, if you want to modify the name attribute of a related Company entity of an Employee entity, you can write:

```

$emp:=ds.Employee.get(2) // load the Employee entity with primary key
2
$emp.employer.name:="4D, Inc." //modify the name attribute of the
related Company
$emp.employer.save() //save the related attribute
//the related entity is updated

```

## Creating an entity selection

You can create an object of type [entity selection](#) as follows:

- Querying the entities [in a dataclass](#) or in an [existing entity selection](#);
- Using the [.all\(\)](#) dataclass function to select all the entities in a dataclass;
- Using the [Create entity selection](#) command or the [.newSelection\(\)](#) dataclass function to create a blank entity selection;
- Using the [.copy\(\)](#) function to duplicate an existing entity selection;
- Using one of the various functions from the [Entity selection class](#) that returns a new entity selection, such as [.or\(\)](#);
- Using a relation attribute of type "related entities" (see below).

You can simultaneously create and use as many different entity selections as you want for a dataclass. Keep in mind that an entity selection only contains references to entities. Different entity selections can contain references to the same entities.

## Shareable or alterable entity selections

An entity selection can be **shareable** (readable by multiple processes, but not

alterable after creation) or **alterable** (supports the `.add()` function, but only usable by the current process).

## Properties

A **shareable** entity selection has the following characteristics:

- it can be stored in a shared object or shared collection, and can be passed as parameter between several processes or workers;
- it can be stored in several shared objects or collections, or in a shared object or collection which already belongs to a group (it does not have a *locking identifier*);
- it does not allow the addition of new entities. Trying to add an entity to a shareable entity selection will trigger an error (1637 - This entity selection cannot be altered). To add an entity to a shareable entity selection, you must first transform it into a non-shareable entity selection using the `.copy()` function, before calling `.add()`.

Most entity selection functions (such as `.slice()`, `.and()`...) support shareable entity selections since they do not need to alter the original entity selection (they return a new one).

An **alterable** entity selection has the following characteristics:

- it cannot be shared between processes, nor be stored in a shared object or collection. Trying to store a non-shareable entity selection in a shared object or collection will trigger an error (-10721 - Not supported value type in a shared object or shared collection);
- it accepts the addition of new entities, i.e. it supports the `.add()` function.

## How are they defined?

The **shareable** or **alterable** nature of an entity selection is defined when the entity selection is created (it cannot be modified afterwards). You can know the nature of an entity selection using the `.isAlterable()` function or the `OB Is shared` command.

A new entity selection is **shareable** in the following cases:

- the new entity selection results from an ORDA class function applied to a dataClass: `dataClass.all()`, `dataClass.fromCollection()`, `dataClass.query()`,
- the new entity selection is based upon a relation `entity.attributeName` (e.g. "company.employees") when `attributeName` is a one-to-many related attribute but the entity does not belong to an entity selection.
- the new entity selection is explicitly copied as shareable with `entitySelection.copy()` or `OB Copy` (i.e. with the `ck shared` option).

Example:

```
$myComp:=ds.Company.get(2) // $myComp does not belong to an entity selection  
$employees:=$myComp.employees // $employees is shareable
```

A new entity selection is **alterable** in the following cases:

- the new entity selection created blank using the `dataClass.newSelection()` function or `Create entity selection` command,
- the new entity selection is explicitly copied as alterable with `entitySelection.copy()` or `OB Copy` (i.e. without the `ck shared` option).

Example:

```
$toModify:=ds.Company.all().copy() // $toModify is alterable
```

A new entity selection **inherits** from the original entity selection nature in the following cases:

- the new entity selection results from one of the various ORDA class functions applied to an existing entity selection ([.query\(\)](#), [.slice\(\)](#), etc.).
- the new entity selection is based upon a relation:
  - [entity.attributeName](#) (e.g. "company.employees") when *attributeName* is a one-to-many related attribute and the entity belongs to an entity selection (same nature as [.getSelection\(\)](#) entity selection),
  - [entitySelection.attributeName](#) (e.g. "employees.employer") when *attributeName* is a related attribute (same nature as the entity selection),
  - [.extract\(\)](#) when the resulting collection contains entity selections (same nature as the entity selection).

Examples:

```
$highSal:=ds.Employee.query("salary >= :1"; 1000000)
    // $highSal is shareable because of the query on dataClass
$comp:=$highSal.employer // $comp is shareable because $highSal is
shareable
```

```
$lowSal:=ds.Employee.query("salary <= :1"; 10000).copy()
    // $lowSal is alterable because of the copy()
$comp2:=$lowSal.employer // $comp2 is alterable because $lowSal is
alterable
```

## ⓘ ENTITY SELECTIONS RETURNED FROM THE SERVER

In client/server architecture, entity selections returned from the server are always shareable on the client, even if [copy\(\)](#) was called on the server. To make such an entity selection alterable on the client, you need to execute [copy\(\)](#) on the client side.

Example:

```
// a function is always executed on the server
exposed Function getSome() : cs.MembersSelection
    return This.query("ID >= :1"; 15).orderBy("ID ASC")

// in a method, executes on the remote side
var $result : cs.MembersSelection
var $alterable : Boolean
$result:=ds.Members.getSome() // $result is shareable
$alterable:=$result.isAlterable() // False

$result:=ds.Members.getSome().copy() // $result is now alterable
$alterable:=$result.isAlterable() // True
```

## Sharing an entity selection between processes (example)

You work with two entity selections that you want to pass to a worker process so that it can send mails to appropriate persons:

```

var $paid; $unpaid : cs.InvoicesSelection
//We get entity selections for paid and unpaid invoices
$paid:=ds.Invoices.query("status=:1"; "Paid")
$unpaid:=ds.Invoices.query("status=:1"; "Unpaid")

//We pass entity selection references as parameters to the worker
CALL WORKER("mailing"; "sendMails"; $paid; $unpaid)

```

The `sendMails` method:

```

#DECLARE ($paid : cs.InvoicesSelection; $unpaid :
cs.InvoicesSelection)
var $invoice : cs.InvoicesEntity

var $server; $transporter; $email; $status : Object

//Prepare emails
$server:=New object()
$server.host:="exchange.company.com"
$server.user:="myName@company.com"
$server.password:="my!!password"
$transporter:=SMTP New transporter($server)
$email:=New object()
$email.from:="myName@company.com"

//Loops on entity selections
For each($invoice;$paid)
    $email.to:=$invoice.customer.address // email address of the
customer
    $email.subject:="Payment OK for invoice # "+String($invoice.number)

    $status:=$transporter.send($email)
End for each

For each($invoice;$unpaid)
    $email.to:=$invoice.customer.address // email address of the
customer
    $email.subject:="Please pay invoice # "+String($invoice.number)
    $status:=$transporter.send($email)
End for each

```

## Entity selections and Storage attributes

All storage attributes (text, number, boolean, date) are available as properties of entity selections as well as entities. When used in conjunction with an entity selection, a scalar attribute returns a collection of scalar values. For example:

```

$locals:=ds.Person.query("city = :1";"San Jose") //entity selection of
people
$localEmails:=$locals.emailAddress //collection of email addresses
(strings)

```

This code returns in `$localEmails` a collection of email addresses as strings.

## Entity selections and Relation attributes

In addition to the variety of ways you can query, you can also use relation attributes as properties of entity selections to return new entity selections. For example, consider the following structure:

```

$myParts:=ds.Part.query("ID < 100") //Return parts with ID less than
100
$myInvoices:=$myParts.invoiceItems.invoice
//All invoices with at least one line item related to a part in
$myParts

```

The last line will return in \$myInvoices an entity selection of all invoices that have at least one invoice item related to a part in the entity selection myParts. When a relation attribute is used as a property of an entity selection, the result is always another entity selection, even if only one entity is returned. When a relation attribute is used as a property of an entity selection and no entities are returned, the result is an empty entity selection, not null.

## Entity Locking

You often need to manage possible conflicts that might arise when several users or processes load and attempt to modify the same entities at the same time. Record locking is a methodology used in relational databases to avoid inconsistent updates to data. The concept is to either lock a record upon read so that no other process can update it, or alternatively, to check when saving a record to verify that some other process hasn't modified it since it was read. The former is referred to as **pessimistic record locking** and it ensures that a modified record can be written at the expense of locking records to other users. The latter is referred to as **optimistic record locking** and it trades the guarantee of write privileges to the record for the flexibility of deciding write privileges only if the record needs to be updated. In pessimistic record locking, the record is locked even if there is no need to update it. In optimistic record locking, the validity of a record's modification is decided at update time.

ORDA provides you with two entity locking modes:

- an automatic "optimistic" mode, suitable for most applications,
- a "pessimistic" mode allowing you to lock entities prior to their access.

### Automatic optimistic lock

This automatic mechanism is based on the concept of "optimistic locking" which is particularly suited to the issues of web applications. This concept is characterized by the following operating principles:

- All entities can always be loaded in read-write; there is no *a priori* "locking" of entities.
- Each entity has an internal locking stamp that is incremented each time it is saved.
- When a user or process tries to save an entity using the `entity.save()` method, 4D compares the stamp value of the entity to be saved with that of the entity found in the data (in the case of a modification):
  - When the values match, the entity is saved and the internal stamp value is incremented.
  - When the values do not match, it means that another user has modified this entity in the meantime. The save is not performed and an error is returned.

The following diagram illustrates optimistic locking:

1. Two processes load the same entity.

2. The first process modifies the entity and validates the change. The `entity.save()` method is called. The 4D engine automatically compares the internal stamp value of the modified entity with that of the entity stored in the data. Since they match, the entity is saved and its stamp value is incremented.
  
3. The second process also modifies the loaded entity and validates its changes. The `entity.save()` method is called. Since the stamp value of the modified entity does not match the one of the entity stored in the data, the save is not performed and an error is returned.

This can also be illustrated by the following code:

```
$person1:=ds.Person.get(1) //Reference to entity
$person2:=ds.Person.get(1) //Other reference to same entity
$person1.name:="Bill"
$result:=$person1.save() //result.success=true, change saved
$person2.name:="William"
$result:=$person2.save() //result.success=false, change not saved
```

In this example, we assign to \$person1 a reference to the person entity with a key of 1. Then, we assign another reference of the same entity to variable \$person2. Using \$person1, we change the first name of the person and save the entity. When we attempt to do the same thing with \$person2, 4D checks to make sure the entity on disk is the same as when the reference in \$person1 was first assigned. Since it isn't the same, it returns false in the success property and doesn't save the second modification.

When this situation occurs, you can, for example, reload the entity from the disk using the `entity.reload()` method so that you can try to make the modification again. The `entity.save()` method also proposes an "automerge" option to save the entity in case processes modified attributes that were not the same.

Record stamps are not used in **transactions** because only a single copy of a record exists in this context. Whatever the number of entities that reference a record, the same copy is modified thus `entity.save()` operations will never generate stamp errors.

## Pessimistic lock

You can lock and unlock entities on demand when accessing data. When an entity is getting locked by a process, it is loaded in read/write in this process but it is locked for all other processes. The entity can only be loaded in read-only mode in these processes; its values cannot be edited or saved.

This feature is based upon two functions of the `Entity` class:

- `entity.lock()`
- `entity.unlock()`

For more information, please refer to the descriptions for these functions.

Pessimistic locks can also be handled through the [REST API](#).

## Concurrent use of 4D classic locks and ORDA pessimistic locks

Using both classic and ORDA commands to lock records is based upon the following principles:

- A lock set with a classic 4D command on a record prevents ORDA to lock the entity matching the record.
- A lock set with ORDA on an entity prevents classic 4D commands to lock the record matching the entity.

These principles are shown in the following diagram:

**Transaction locks** also apply to both classic and ORDA commands. In a multiprocess or a multi-user application, a lock set within a transaction on a record by a classic command will result in preventing any other processes to lock entities related to this record (or conversely), until the transaction is validated or canceled.

- Example with a lock set by a classic command:
- Example with a lock set by an ORDA function:

## Using a remote datastore

A [datastore](#) exposed on a 4D application can be accessed simultaneously through different clients:

- 4D remote applications using ORDA to access the main datastore with the `ds` command. Note that the 4D remote application can still access the database in classic mode. These accesses are handled by the **4D application server**.
- Other 4D applications (4D remote, 4D Server) opening a session on the remote datastore through the [Open datastore](#) command. These accesses are handled by the **HTTP REST server**.
- [4D for iOS or 4D for Android](#) queries for updating mobile applications. These accesses are handled by the **HTTP server**.

## Opening sessions

When you work with a remote datastore referenced through calls to the [Open datastore](#) command, the connection between the requesting processes and the remote datastore is handled via sessions.

A session is created on the remote datastore to handle the connection. This session is identified using a internal session ID which is associated to the `localID` on the 4D application side. This session automatically manages access to data, entity selections, or entities.

The `localID` is local to the machine that connects to the remote datastore, which means:

- If other processes of the same application need to access the same remote datastore, they can use the same `localID` and thus, share the same session.
- If another process of the same application opens the same remote datastore but with another `localID`, it will create a new session on the remote datastore.
- If another machine connects to the same remote datastore with the same `localID`, it will create another session with another cookie.

These principles are illustrated in the following graphics:

For sessions opened by REST requests, please refer to [Users and sessions](#).

## Viewing sessions

Processes that manage sessions for datastore access are shown in the 4D Server administration window:

- name: "REST Handler: ¥<process name>"
- type: HTTP Server Worker type
- session: session name is the user name passed to the [Open datastore](#) command.

In the following example, two processes are running for the same session:

## Locking and transactions

ORDA features related to entity locking and transaction are managed at process level in remote datastores, just like in ORDA client/server mode:

- If a process locks an entity from a remote datastore, the entity is locked for all other processes, even when these processes share the same session (see [Entity locking](#)). If several entities pointing to a same record have been locked in a process, they must be all unlocked in the process to remove the lock. If a lock has been put on an entity, the lock is removed when there is no more reference to this entity in memory.
- Transactions can be started, validated or cancelled separately on each remote datastore using the `dataStore.startTransaction()`, `dataStore.cancelTransaction()`, and `dataStore.validateTransaction()` functions. They do not impact other datastores.
- Classic 4D language commands (`START TRANSACTION`, `VALIDATE TRANSACTION`, `CANCEL TRANSACTION`) only apply to the main datastore (returned by `ds`). If an entity from a remote datastore is hold by a transaction in a process, other processes cannot update it, even if these processes share the same session.
- Locks on entities are removed and transactions are rolled back:
  - when the process is killed.
  - when the session is closed on the server
  - when the session is killed from the server administration window.

## Closing sessions

A session is automatically closed by 4D when there has been no activity during its timeout period. The default timeout is 60 mn, but this value can be modified using the `connectionInfo` parameter of the `Open datastore` command.

If a request is sent to the remote datastore after the session has been closed, it is automatically re-created if possible (license available, server not stopped...). However, keep in mind that the context of the session regarding locks and transactions is lost (see above).

## Client/server optimization

4D provides optimizations for ORDA requests that use entity selections or load entities in client/server configurations (datastore accessed remotely through `ds` or via `Open datastore`). These optimizations speed up the execution of your 4D application by reducing drastically the volume of information transmitted over the network. They include:

- the **optimization context**
- the **ORDA cache**

### Context

The optimization context is based upon the following implementations:

- When a client requests an entity selection from the server, 4D automatically "learns" which attributes of the entity selection are actually used on the client side during the code execution, and builds a corresponding "optimization context". This context is attached to the entity selection and stores the used attributes. It will be dynamically updated if other attributes are used afterwards. The following methods and functions trigger the learning phase:
  - [Create entity selection](#)
  - [dataClass.fromCollection\(\)](#)

- [dataClass.all\(\)](#)
- [dataClass.get\(\)](#)
- [dataClass.query\(\)](#)
- [entitySelection.query\(\)](#)

- Subsequent requests sent to the server on the same entity selection automatically reuse the optimization context and only get necessary attributes from the server, which accelerates the processing. For example, in an [entity selection-based list box](#), the learning phase takes place during the display of the first row. the display of the next rows is optimized. The following functions automatically associate the optimization context of the source entity selection to the returned entity selection:

- [entitySelection.and\(\)](#)
- [entitySelection.minus\(\)](#)
- [entitySelection.or\(\)](#)
- [entitySelection.orderBy\(\)](#)
- [entitySelection.slice\(\)](#)
- [entitySelection.drop\(\)](#)

- An existing optimization context can be passed as a property to another entity selection of the same dataclass, thus bypassing the learning phase and accelerating the application (see [Using the context property](#) below).
- You can build optimization contexts manually using the [dataStore.setRemoteContextInfo\(\)](#) function (see [Preconfiguring contexts](#)).

## Example

Given the following code:

```
$sel:=$ds.Employee.query("firstname = ab@")
For each($e;$sel)
    $s:=$e.firstname+" "+$e.lastname+ " works for "+$e.employer.name // 
$e.employer refers to Company table
End for each
```

Thanks to the optimization, this request will only get data from used attributes (firstname, lastname, employer, employer.name) in \$sel from the second iteration of the loop.

## Reusing the context property

You can increase the benefits of the optimization by using the **context** property. This property references an optimization context "learned" for an entity selection. It can be passed as parameter to ORDA functions that return new entity selections, so that entity selections directly request used attributes to the server and bypass the learning phase.

You can also create contexts using the [.setRemoteContextInfo\(\)](#) function.

The same optimization context property can be passed to unlimited number of entity selections on the same dataclass. All ORDA functions that handle entity selections support the **context** property (for example [dataClass.query\(\)](#) or [dataClass.all\(\)](#)). Keep in mind, however, that a context is automatically updated when new attributes are used in other parts of the code. Reusing the same context in

different codes could result in overloading the context and then, reduce its efficiency.

A similar mechanism is implemented for entities that are loaded, so that only used attributes are requested (see the [dataClass.get\(\)](#) function).

### Example with `dataClass.query()`:

```
var $sel1; $sel2; $sel3; $sel4; $querysettings; $querysettings2 :  
Object  
var $data : Collection  
$querysettings:=New object("context";"shortList")  
$querysettings2:=New object("context";"longList")  
  
$sel1:=ds.Employee.query("lastname = S@";$querysettings)  
$data:=extractData($sel1) // In extractData method an optimization is  
triggered  
// and associated to context "shortList"  
  
$sel2:=ds.Employee.query("lastname = Sm@";$querysettings)  
$data:=extractData($sel2) // In extractData method the optimization  
associated  
// to context "shortList" is applied  
  
$sel3:=ds.Employee.query("lastname = Smith";$querysettings2)  
$data:=extractDetailedData($sel3) // In extractDetailedData method an  
optimization  
// is triggered and associated to context "longList"  
  
$sel4:=ds.Employee.query("lastname = Brown";$querysettings2)  
$data:=extractDetailedData($sel4) // In extractDetailedData method the  
optimization  
// associated to context "longList" is applied
```

## Entity selection-based list box

Entity selection optimization is automatically applied to entity selection-based list boxes in client/server configurations, when displaying and scrolling a list box content: only the attributes displayed in the list box are requested from the server.

A specific "page mode" context is also provided when loading the current entity through the **Current item** property expression of the list box (see [Collection or entity selection type list boxes](#)). This feature allows you to not overload the list box initial context in this case, especially if the "page" requests additional attributes. Note that only the use of **Current item** expression will create/use the page context (access through `entitySelection\[index]` will alter the entity selection context).

Subsequent requests to server sent by entity browsing functions will also support this optimization. The following functions automatically associate the optimization context of the source entity to the returned entity:

- [entity.next\(\)](#)
- [entity.first\(\)](#)
- [entity.last\(\)](#)
- [entity.previous\(\)](#)

For example, the following code loads the selected entity and allows browsing in the entity selection. Entities are loaded in a separate context and the list box initial context is left untouched:

```
$myEntity:=Form.currentElement //current item expression  
//... do something  
$myEntity:=$myEntity.next() //loads the next entity using the same context
```

## Preconfiguring contexts

An optimization context should be defined for every feature or algorithm of your application, in order to have the best performances. For example, a context can be used for queries on customers, another context for queries on products, etc.

If you want to deliver final applications with the highest level of optimization, you can preconfigure your contexts and thus save learning phases by following these steps:

1. Design your algorithms.
2. Run your application and let the automatic learning mechanism fill the optimization contexts.
3. Call the [dataStore.getRemoteContextInfo\(\)](#) or [dataStore.getAllRemoteContexts\(\)](#) function to collect contexts. You can use the [entitySelection.getRemoteContextAttributes\(\)](#) and [entity.getRemoteContextAttributes\(\)](#) functions to analyse how your algorithms use attributes.
4. In the final step, call the [datastore.setRemoteContextInfo\(\)](#) function to build contexts at application startup and [use them](#) in your algorithms.

## ORDA cache

For optimization reasons, data requested from the server via ORDA is loaded in the ORDA remote cache (which is different from the 4D cache). The ORDA cache is organized by dataclass, and expires after 30 seconds.

The data contained in the cache is considered as expired when the timeout is reached. Any access to expired data will send a request to the server. Expired data remains in the cache until space is needed.

By default, the ORDA cache is transparently handled by 4D. However, you can control its contents using the following ORDA class functions:

- [dataClass.setRemoteCacheSettings\(\)](#)
- [dataClass.getRemoteCache\(\)](#)
- [dataClass.clearRemoteCache\(\)](#)

# Privileges

Protecting data while allowing fast and easy access to authorized users is a major challenge for web applications. The ORDA security architecture is implemented at the heart of your datastore and allows you to define specific privileges to all user sessions for the various resources in your project (datastore, dataclasses, functions, etc.).

## Overview

The ORDA security architecture is based upon the concepts of privileges, permission actions (read, create, etc.), and resources.

When users get logged, their session is automatically loaded with associated privilege(s). Privileges are assigned to the session using the [`session.setPrivileges\(\)`](#) function.

Every user request sent within the session is evaluated against privileges defined in the project's `roles.json` file.

If a user attempts to execute an action and does not have the appropriate access rights, a privilege error is generated or, in the case of missing Read permission on attributes, they are not sent.

## Resources

You can assign specific permission actions to the following exposed resources in your project:

- the datastore
- a dataclass
- an attribute (including computed and alias)
- a data model class function

A permission action defined at a given level is inherited by default at lower levels, but several permissions can be set:

- A permission action defined at the datastore level is automatically assigned to all dataclasses.
- A permission action defined at a dataclass level overrides the datastore setting (if any). By default, all attributes of the dataclass inherit from the dataclass permission(s).
- Unlike dataclass permissions, a permission action defined at the attribute level does not override the parent dataclass permission(s), but is added to. For example, if you assigned the "general" privilege to a dataclass and the "detail" privilege to an attribute of the dataclass, both "general" and "detail" privileges must be set to the session to access the attribute.

## Permission actions

Available actions are related to target resource.

| <b>Actions</b>  | <b>datastore</b>   | <b>dataclass</b>  | <b>attribute</b>  | <b>data model function</b>   |
|-----------------|--|---|---|--|
| <b>create</b>   | Create entity in any dataclass   | Create entity in this dataclass   | Create an entity with a value different from default value allowed for this attribute (ignored for alias attributes). | n/a  |
| <b>read</b>     | Read attributes in any dataclass   | Read attributes in this dataclass   | Read this attribute content   | n/a  |
| <b>update</b>   | Update attributes in any dataclass.  | Update attributes in this dataclass.  | Update this attribute content (ignored for alias attributes).   | n/a  |
| <b>drop</b>     | Delete data in any dataclass.  | Delete data in this dataclass.  | Delete a not null value for this attribute (except for alias and computed attribute).                                 | n/a  |
| <b>execute</b>  | Execute any function on the project (datastore, dataclass, entity selection, entity) | Execute any function on the dataclass. Dataclass functions, entity functions, and entity selection functions are handled as dataclass functions | n/a   | Execute this function  |
| <b>describe</b> | All the dataclasses are available in the /rest/\$catalog API                         | This dataclass is available in the /rest/\$catalog API  | This attribute is available in the /rest/\$catalog API.   | This dataclass function is available in the /rest/\$catalog API.   |
| <b>promote</b>  | n/a  | n/a   | n/a   | Associates a given privilege during the execution of the function. The privilege is temporary added to the session and removed at the end of the function execution. By security, only the process executing the function is added the privilege, not the whole session. |

**Notes:**

- An alias can be read as soon as the session privileges allow the access to the alias itself, even if the session privileges do no allow the access to the attributes resolving the alias.
- A computed attribute can be accessed even if there are no permissions on the attributes upon which it is built.
- Default values: in the current implementation, only *Null* is available as default value.

Setting permissions requires to be consistent, in particular:

- **update** and **drop** permissions also need **read** permission (but **create** does not need it)
- **promote** permission also need **describe** permission.

## Privileges and Roles

A **privilege** is the technical ability to run **actions** on **resources**, while a **role** is a privilege published to be used by an administrator. Basically, a role gathers several privileges to define a business user profile. For example, "manageInvoices" could be a privilege while "secretary" could be a role (which includes "manageInvoices" and other privileges).

A privilege or a role can be associated to several "action + resource" combinations. Several privileges can be associated to an action. A privilege can include other privileges.

- You **create** privileges and/or roles in the `roles.json` file (see below). You **configure** their scope by assigning them to permission action(s) applied to resource(s).
- You **allow** privileges and/or roles to every user session using the `.setPrivileges\(\)` function of the `Session` class.

## Example

To allow a role in a session:

```
exposed Function authenticate($identifier : Text; $password : Text) ->$result : Text

    var $user : cs.UsersEntity

    Session.clearPrivileges()

    $result:="Your are authenticated as Guest"

    $user:=ds.Users.query("identifier = :1"; $identifier).first()

    If ($user#Null)
        If (Verify password hash($password; $user.password))
            Session.setPrivileges(New object("roles"; $user.role))
            $result:="Your are authenticated as "+$user.role
        End if
    End if
```

## `roles.json` file

The `roles.json` file describes the whole security settings for the project.

### ⓘ NOTE

In a context other than *Qodly* (cloud), you have to create this file at the following location: `<project folder>/Project/Sources/`. See [Architecture](#) section.

The `roles.json` file syntax is the following:

| Property name | Type  | Mandatory             | Description   |
|---------------|---|-----------------------|---|
| privileges    | Collection of <code>privilege</code> objects  | X                     | List of defined privileges  |
|               | <code>[]</code> .privilege                    |                       | Privilege name  |
|               | <code>[]</code> .includes                     |                       | List of included privilege names  |
| roles         | Collection of <code>role</code> objects       |                       | List of defined roles   |
|               | <code>[]</code> .role                         |                       | Role name   |
|               | <code>[]</code> .privileges                   |                       | List of included privilege names  |
| permissions   | Object  | X                     | List of allowed actions   |
|               | Collection of <code>permission</code> objects |                       | List of allowed permissions   |
|               | <code>[]</code> .applyTo                      |                       | Targeted <a href="#">resource</a> name  |
| allowed       | <code>[]</code> .type                         | String                | <a href="#">Resource</a> type:<br>"datastore",<br>"dataclass", "attribute",<br>"method" |
|               | <code>[]</code> .read                         | Collection of strings | List of privileges  |
|               | <code>[]</code> .create                       | Collection of strings | List of privileges  |
|               | <code>[]</code> .update                       | Collection of strings | List of privileges  |
|               | <code>[]</code> .drop                         | Collection of strings | List of privileges  |
|               | <code>[]</code> .describe                     | Collection of strings | List of privileges  |
|               | <code>[]</code> .execute                      | Collection of strings | List of privileges  |
|               | <code>[]</code> .promote                      | Collection of strings | List of privileges  |

### ⚠ REMINDER

- The "WebAdmin" privilege name is reserved to the application. It is not recommended to use this name for custom privileges.
- `privileges` and `roles` names are case insensitive.

### `Roles_Errors.json` file

The `roles.json` file is parsed by 4D at startup. You need to restart the application if

you want modifications in this file to be taken into account.

In case of error(s) when parsing the `roles.json` file, 4D loads the project but disables the global access protection - this allows the developer to access the files and to fix the error. An error file named `Roles_Errors.json` is generated in the [Logs folder of the project](#) and describes the error line(s). This file is automatically deleted when the `roles.json` file no longer contains error(s).

It is recommended to check at startup if a `Roles_Errors.json` file exists in the [Logs folder](#), which means that there was a parsing error and that accesses will not be limited. You can write for example:

```
/Sources/DatabaseMethods/onStartup.4dm
If (Not (File ("/LOGS/"+"Roles_Errors.json").exists))
...
Else // you can prevent the project to open
    ALERT("The roles.json file is malformed or contains inconsistencies,
the application will quit.")
    QUIT 4D
End if
```

## Initializing privileges for deployment

By default, if no specific parameters are defined in the `roles.json` file, accesses are not limited. This configuration allows you to develop the application without having to worry about accesses.

However, when the application is about to be deployed, a good practice is to lock all privileges and then, to configure the file to only open controlled parts to authorized sessions. To lock all privileges on all resources, put the following `roles.json` file in your project folder (it includes examples of methods):

```
/Project/Sources/roles.json
{
    "privileges": [
        {
            "privilege": "none",
            "includes": []
        }
    ],
    "roles": [],
    "permissions": {
        "allowed": [
            {
                "applyTo": "ds",
                "type": "datastore",
                "read": [
                    "none"
                ],
                "create": [
                    "none"
                ],
                "update": [
                    "none"
                ],
                "drop": [
                    "none"
                ],
                "execute": [
                    "none"
                ],
                "databases": [
                    "none"
                ]
            }
        ],
        "denied": [
            {
                "applyTo": "ds",
                "type": "datastore",
                "read": [
                    "none"
                ],
                "create": [
                    "none"
                ],
                "update": [
                    "none"
                ],
                "drop": [
                    "none"
                ],
                "execute": [
                    "none"
                ],
                "databases": [
                    "none"
                ]
            }
        ]
    }
}
```

```
        "describe": [
            "none"
        ],
        "promote": [
            "none"
        ]
    },
    {
        "applyTo": "ds.loginAs",
        "type": "method",
        "execute": [
            "guest"
        ]
    },
    {
        "applyTo": "ds.hasPrivilege",
        "type": "method",
        "execute": [
            "guest"
        ]
    },
    {
        "applyTo": "ds.clearPrivileges",
        "type": "method",
        "execute": [
            "guest"
        ]
    },
    {
        "applyTo": "ds.isGuest",
        "type": "method",
        "execute": [
            "guest"
        ]
    },
    {
        "applyTo": "ds.getPrivileges",
        "type": "method",
        "execute": [
            "guest"
        ]
    },
    {
        "applyTo": "ds.setAllPrivileges",
        "type": "method",
        "execute": [
            "guest"
        ]
    }
}
```

# Glossary

## Main concepts at a glance

### Action

Every action that can be done on a [resource](#). Available actions are: create, read, update, drop, execute, promote, and describe.

### Attribute

An attribute is the smallest storage cell in a relational database (see also [Relation attribute](#)). Do not confuse dataclass attributes and entity attributes:

- In a dataclass object, each property is a dataclass attribute that maps to a corresponding field in the corresponding table (same name and type).
- In an entity object, entity attributes are properties that contain values for the corresponding datastore attributes.

*Attributes* and *properties* are similar concepts. "Attribute" is used to designate dataclass properties that store data, while "property" is more generic and defines a piece of data stored within an object.

### AttributePath

An attributePath is the path of an attribute inside a given dataclass or entity. See also [PropertyPath](#).

### Class code

Code for the user class function(s).

### Computed attribute

A computed attribute doesn't actually store information. Instead, it determines its value based on other values from the same entity or from other entities, attributes or functions. When a computed attribute is referenced, the underlying "computation" is evaluated to determine the value. Computed attributes may even be assigned values where user-defined code determines what to do during the assignment.

### Data model class

Extended class available for a data model object.

### Data model object

Database objects available through the ORDA concept, i.e. datastore, dataclasses, entities and entity selections.

### Data model function

Function of an ORDA data model class.

## Dataclass

A dataclass is an object model that describes the data. Tables in the database provided by the datastore are handled through dataclasses. Each table in the database provided by the datastore has a corresponding dataclass with the same name. Each field of the table is an attribute of the dataclass.

A dataclass is related to a single datastore.

## DataClass class

Class for specific dataclass objects, in which you can add custom functions.

## Datastore

A datastore is the interface object provided by ORDA to reference a structure and access its data. The main database, returned by the `ds` command, is available as a datastore (the main datastore).

A datastore provides:

- a connection to the 4D database
- a set of dataclasses to work with the database

The database can be a 4D local database (the Main datastore), or a 4D Server database exposed as REST resource (a Remote datastore).

A datastore references only a single database. It is, however, possible to open several datastores to access several databases.

## DataStore class

Class for datastore objects, in which you can add custom functions.

## DataStoreImplementation

Internal name of the generic DataStore class in the `4D` class store.

## Deep copy

A deep copy duplicates an object and all the references it contains. After a deep copy, a copied collection contains duplicated elements and thus, new references, of all of the orginal elements. See also Shallow copy.

## ds

`ds` is the 4D language command that returns a [datastore](#) object reference. It matches the datastore available upon the 4D main database.

## Entity

An entity is an object that corresponds to a dataclass model. An entity contains the same attributes as the dataclass.

An entity can be seen as an instance of the dataclass, like a record of the table matching the dataclass in its associated datastore. However, an entity also contains related data. The purpose of the entity is to manage data (create, update, delete).

For more information, see Entities.

## Entity selection

An entity selection is an object. When querying the datastore, an entity selection is returned. An entity selection is a set of references to entities related to the same dataclass.

An entity selection contains:

- a set of 0 to X entity references,
- a length property (always),
- queryPlan and queryPath properties (if asked while querying).

An entity selection can also be empty.

## Generic class

Built-in class for ORDA objects such as entities, or dataclasses. Functions and properties of generic classes are automatically available in user extended classes, e.g. `EmployeeEntity`.

## Lazy loading

Since entities are managed as references, data is loaded only when necessary, i.e. when accessing it in the code or through interface widgets. This optimization principle is called lazy loading.

## Main datastore

The Datastore object matching the opened 4D database (standalone or client/server). The main datastore is returned by the `ds` command.

## Method

ORDA objects such as datastores, dataclasses, entity selections, and entities, define classes of objects. They provide specific methods to directly interact with them. These methods are also called member functions. Such methods are used by calling them on an instance of the object.

For example, the `query()` method is a dataclass member function. If you have stored a dataclass object in the `$myClass` variable, you can write:

```
$myClass.query("name = smith")
```

## Mixed data type

In this documentation, "Mixed" data type is used to designate the various type of values that can be stored within dataclass attributes. It includes:

- number
- text
- null
- boolean
- date
- object
- collection
- picture(\*)

(\*) picture type is not supported by statistical methods such as  
entitySelection.max( ).

## Optimistic Lock

In "optimistic lock" mode, entities are not locked explicitly before updating them. Each entity has an internal stamp that is automatically incremented each time the entity is saved on disk. The entity.save( ) or entity.drop( ) methods will return an error if the stamp of the loaded entity (in memory) and the stamp of the entity on disk do not match, or if the entity has been dropped. Optimistic locking is only available in ORDA implementation. See also "Pessimistic lock".

## Pessimistic Lock

A "pessimistic lock" means that an entity is locked prior to its being accessed, using the entity.lock( ) method. Other processes can neither update nor drop the entity until it is unlocked. The classic 4D language only allows pessimistic locks. See "Optimistic lock".

## Privilege

The ability to run one or more [actions](#) on [resources](#). Several privileges can be gathered in a [role](#) according to the business logic.

## Property

See [Attribute](#).

Attributes and properties are similar concepts. "Attribute" is used to designate dataclass properties that store data, while "property" is more generic and defines a piece of data stored within an object.

## PropertyPath

A propertyPath is the path to a property in a given object. If the property is nested in several levels, each level separated is by a dot (".").

## Regular class

User class not related to an ORDA object.

## Related dataclass

These are dataclasses linked by relation attributes.

## Relation attribute

Relation attributes are used to conceptualize relations between dataclasses (many-to-one and one-to-many).

- Many-to-one relation (dataclassA references an occurrence of dataclassB): a relation attribute is available in dataclassA and references one instance of dataclassB.
- One-to-many relation (an occurrence of dataclassB references several occurrences of dataclassA): a relation attribute is available in dataclassB and references several instances of dataclassA.

A dataclass can have recursive relation attributes.

In an entity, the value of a relation attribute can be an entity or an entity selection.

## Related entities

A related entity can be seen as the instance of a relation attribute in a dataclass.

Entity selections may refer to related entities according to the relation attributes defined in the corresponding dataclasses.

## Remote datastore

A 4D database opened on a 4D or 4D Server (available through HTTP) and exposed as a REST resource. This database can be referenced locally as a Datastore from other workstations, where it is assigned a locaID. The remote datastore can be used through ORDA concepts (datastore, dataclass, entity selection...). This use is submitted to a licencing system.

## Resource

An ORDA element on which any [action](#) can be allowed or not according to a [privilege](#). Available resources are: the datastore, a dataclass, a dataclass attribute, an ORDA Data model function, or a project method.

## Role

A role is a published [privilege](#) intended to be used by an administrator. It can contain one or more privileges.

## Session

When the 4D application connects to a Remote datastore, a session is created on the 4D Server (HTTP). A session cookie is generated and associated to the local datastore id.

Each time a new session is opened, a license is used. Each time a session is closed, the license is freed.

Inactive sessions are automatically closed after a timeout. The default timeout is 48 hours, it can be set by the developer (it must be  $\geq 60$  minutes).

## Shallow copy

A shallow copy only duplicates the structure of elements, and keeps the same internal references. After a shallow copy, two collections will both share the individual elements. See also Deep copy.

## Stamp

Used in "optimistic" locking technology. All entities have an internal counter, the stamp, which is incremented each time the entity is saved. By automatically comparing stamps between an entity being saved and its version stored on disk, 4D can prevent concurrent modifications on the same entities.

## Storage attribute

A storage attribute (sometimes referred to as a scalar attribute) is the most basic type

of attribute in a datastore class and most directly corresponds to a field in a relational database. A storage attribute holds a single value for each entity in the class.

# Class Functions

List of built-in 4D classes

## **About class functions**

This section describes the built-in 4D class functions as well as the associated constru...

## **Blob**

The Blob class lets you create and manipulate blob objects (4D.Blob).

## **Class**

When a user class is defined in the project, it is loaded in the 4D language environme...

## **Collection**

The Collection class manages Collection type variables.

## **CryptoKey**

The CryptoKey class in the 4D language encapsulates an asymmetric encryption key ...

## **DataClass**

A DataClass provides an object interface to a database table. All dataclasses in a 4D ...

## **DataClassAttribute**

Dataclass attributes are available as properties of their respective classes. For examp...

## **DataStore**

A Datastore is the interface object provided by ORDA to reference and access a database.

## **Email**

Creating, sending or receiving emails in 4D is done by handling an Email object.

## **Entity**

An entity is an instance of a Dataclass, like a record of the table matching the dataclass definition.

## **EntitySelection**

An entity selection is an object containing one or more reference(s) to entities belonging to a specific class.

## **File**

File objects are created with the File command. They contain references to disk files.

## **FileHandle**

The FileHandle class has functions that allow you to sequentially read from or append to a file.

## **Folder**

Folder objects are created with the Folder command. They contain references to folder objects.

## **Function**

A 4D.Function object contains a piece of code that can be executed from an object, either directly or via a trigger.

## **HTTPRequest**

The [HTTPRequest](#) class allows you to handle [HTTPRequest](#) objects that can be used t...

## **IMAPTransporter**

The [IMAPTransporter](#) class allows you to retrieve messages from a [IMAP](#) email server.

## **MailAttachment**

Attachment objects allow referencing files within a [Email](#) object. Attachment objects ...

## **POP3Transporter**

The [POP3Transporter](#) class allows you to retrieve messages from a [POP3](#) email server.

## **Session**

Session objects are returned by the [Session](#) command when scalable sessions are en...

## **Signal**

Signals are tools provided by the 4D language to manage interactions and avoid conf...

## **SMTPTransporter**

The [SMTPTransporter](#) class allows you to configure [SMTP](#) connections and send email...

## **SystemWorker**

System workers allow the 4D code to call any external process (a shell command, PH...

## [WebServer](#)

The WebServer class API allows you to start and monitor a web server for the main (…

## [WebSocketConnection](#)

[History](#)

## [WebSocketServer](#)

[History](#)

## [ZIPArchive](#)

A 4D ZIP archive is a File or Folder object containing one or more files or folders, whi…

## [ZIPFile](#)

The following properties and functions from the File class are available to ZIPFile obje…

## [ZIPFolder](#)

The following properties and functions from the Folder class are available to ZIPFolde…

## About class functions

This section describes the built-in 4D class functions as well as the associated constructor commands. 4D class functions and properties are available through class instance objects.

- functions must be called on instances with the `()` operator. For example, `collection.sort()`.
- properties are accessed without parentheses, for example `file.creationTime`. You can also use the `[]` syntax, for example `file["creationTime"]`.
- commands can be called independantly, with or without parameters. For example `Folder(fk database folder)`.

## Writing conventions

The following conventions are used in the function syntax:

- the `{ }` characters (braces) indicate optional parameters. For example, `.delete({ option : Integer } )` means that the *option* parameter may be omitted when calling the function.
- the `{ ; ...param }` notation indicates an unlimited number of parameters. For example, `.concat( value : any { ;...valueN } ) : Collection` means that an unlimited number of values of any type can be passed to the function.
- the `any` keyword is used for parameters that can be of any type that can be stored within attributes (number, text, boolean, date, time, object, collection...).

# Blob

The Blob class lets you create and manipulate [blob objects](#) (`4D.Blob`).

## Summary

[\*\*4D.Blob.new\(\)\*\* : 4D.Blob](#)

[\*\*4D.Blob.new\( blobScal : Blob \)\*\* : 4D.Blob](#)

[\*\*4D.Blob.new\( blobObj : 4D.Blob \)\*\* : 4D.Blob](#) creates a new `4D.Blob` object optionally encapsulating a copy of the data from another blob (scalar blob or `4D.Blob`)

[\*\*.size\*\* : Real](#) returns the size of a `4D.Blob`, expressed in bytes.

[\*\*.slice\(\)\*\* : 4D.Blob](#)

[\*\*.slice\( start : Real \)\*\* : 4D.Blob](#)

[\*\*.slice\( start : Real; end : Real \)\*\* : 4D.Blob](#) creates and returns a `4D.Blob` that references data from a subset of the blob on which it's called. The original blob is not altered.

## 4D.Blob.new()

- History

[\*\*4D.Blob.new\(\)\*\* : 4D.Blob](#)

[\*\*4D.Blob.new\( blobScal : Blob \)\*\* : 4D.Blob](#)

[\*\*4D.Blob.new\( blobObj : 4D.Blob \)\*\* : 4D.Blob](#)

**Parameter      Type      Description**

blob      Blob or `4D.Blob->Blob` to copy

Result      4D.Blob      <- New 4D.Blob

### Description

`4D.Blob.new` creates a new `4D.Blob` object optionally encapsulating a copy of the data from another blob (scalar blob or `4D.Blob`).

If the `blob` parameter is omitted, the method returns an empty 4D.Blob.

## .size

[\*\*.size\*\* : Real](#)

### Description

The `.size` property returns the size of a `4D.Blob`, expressed in bytes.

## .slice()

- History

[\*\*.slice\(\)\*\* : 4D.Blob](#)

[\*\*.slice\( start : Real \)\*\* : 4D.Blob](#)

[\*\*.slice\( start : Real; end : Real \)\*\* : 4D.Blob](#)

| Parameter | Type    | Description   |
|-----------|---------|---|
| start     | Real    | -> index of the first byte to include in the new 4D.Blob.               |
| end       | Real    | -> index of the first byte that will not be included in the new 4D.Blob |
| Result    | 4D.Blob | <- New 4D.Blob  |

## Description

`.slice()` creates and returns a `4D.Blob` that references data from a subset of the blob on which it's called. The original blob is not altered.

The `start` parameter is an index into the blob indicating the first byte to include in the new `4D.Blob`. If you specify a negative value, 4D treats it as an offset from the end of the blob toward the beginning. For example, -10 would be the 10th from last byte in the blob. The default value is 0. If you specify a value for start that is larger than the size of the source blob, the returned `4D.Blob`'s size is 0, and it contains no data.

The `end` parameter is an index into the blob indicating the first byte that will not be included in the new `4D.Blob` (i.e. the byte exactly at this index is not included). If you specify a negative value, 4D treats it as an offset from the end of the blob toward the beginning. For example, -10 would be the 10th from last byte in the blob. The default value is the size of the blob.

## Example

```
var $myBlob : 4D.Blob

// Store text in a 4D.Blob
CONVERT FROM TEXT("Hello, World!"); "UTF-8"; $myBlob)
$isValid:=OB Instance of($myBlob; 4D.Blob); //True

$myString:=Convert to text($myBlob; "UTF-8")
// $myString contains "Hello, World!"

// Create a new 4D.Blob from $myBlob
$myNewBlob:=$myBlob.slice(0; 5)

$myString:=Convert to text($myNewBlob; "UTF-8")
// $myString contains "Hello"
```

# Class

When a user class is [defined](#) in the project, it is loaded in the 4D language environment. A class is an object itself, of "Class" class, which has properties and a function.

## Summary

[\*\*.name : Text\*\*](#) contains the name of the `4D.Class` object

[\*\*.new\( param : any { ;...paramN } \) : 4D.Class\*\*](#) creates and returns a `cs.className` object which is a new instance of the class on which it is called

[\*\*.superclass : 4D.Class\*\*](#) returns the parent class of the class

## .name

- History

**.name : Text**

### Description

The `.name` property contains the name of the `4D.Class` object. Class names are case sensitive.

This property is **read-only**.

## .new()

- History

**.new( param : any { ;...paramN } ) : 4D.Class**

| Parameter | Type     | Description   |
|-----------|----------|---|
| param     | any      | -> Parameter(s) to pass to the constructor function |
| Result    | 4D.Class | <- New object of the class                          |

### Description

The `.new()` function creates and returns a `cs.className` object which is a new instance of the class on which it is called. This function is automatically available on all classes from the [class store](#).

You can pass one or more optional *param* parameters, which will be passed to the [class constructor](#) function (if any) in the `className` class definition. Within the constructor function, the `This` is bound to the new object being constructed.

If `.new()` is called on a non-existing class, an error is returned.

## Examples

To create a new instance of the Person class:

```
var $person : cs.Person
$person:=cs.Person.new() //create the new instance
//$person contains functions of the class
```

To create a new instance of the Person class with parameters:

```
//Class: Person.4dm
Class constructor($firstname : Text; $lastname : Text; $age : Integer)
    This.firstName:=$firstname
    This.lastName:=$lastname
    This.age:=$age
//In a method
var $person : cs.Person
$person:=cs.Person.new("John"; "Doe"; 40)
//$person.firstName = "John"
//$person.lastName = "Doe"
//$person.age = 40
```

## .superclass

- History

**.superclass** : 4D.Class

### Description

The `.superclass` property returns the parent class of the class. A superclass can be a `4D.Class` object, or a `cs.className` object. If the class does not have a parent class, the property returns `null`.

A superclass of a user class is declared in a class by using the [Class extends <superclass>](#) keyword.

This property is **read-only**.

### Examples

```
$sup:=4D.File.superclass //Document
$sup:=4D.Document.superclass //Object
$sup:=4D.Object.superclass //null

// If you created a MyFile class
// with `Class extends File`
$sup:=cs.MyFile.superclass //File
```

See also: [Super](#)

# Collection

The Collection class manages [Collection](#) type variables.

A collection is initialized with:

[\*\*New collection\*\*](#) `{( ...value : any )} : Collection` creates a new empty or prefilled collection

[\*\*New shared collection\*\*](#) `{( ...value : any )} : Collection` creates a new empty or prefilled shared collection

## Example

```
var $colVar : Collection //creation of collection type 4D variable  
$colVar:=New collection //initialization of the collection and  
assignment to the 4D variable
```

## Summary

[\*\*.at\( index : Integer \) : any\*\*](#) returns the item at position *index*, allowing for positive and negative integers

[\*\*.average\( {propertyPath : Text } \) : Real\*\*](#) returns the arithmetic mean (average) of defined values in the collection instance

[\*\*.clear\(\) : Collection\*\*](#) removes all elements from the collection instance and returns an empty collection

[\*\*.combine\( col2 : Collection {; index : Integer } \) : Collection\*\*](#) inserts *col2* elements at the end or at the specified *index* position in the collection instance and returns the edited collection

[\*\*.concat\( value : any { ;...valueN } \) : Collection\*\*](#) returns a new collection containing the elements of the original collection with all elements of the *value* parameter added to the end

[\*\*.copy\(\) : Collection\*\*](#)

[\*\*.copy\( option : Integer \) : Collection\*\*](#)

[\*\*.copy\( option : Integer ; groupWithCol : Collection \) : Collection\*\*](#)

[\*\*.copy\( option : Integer ; groupWithObj : Object \) : Collection\*\*](#) returns a deep copy of the collection instance

[\*\*.count\( { propertyPath : Text } \) : Real\*\*](#) returns the number of non-null elements in the collection

[\*\*.countValues\( value : any {; propertyPath : Text } \) : Real\*\*](#) returns the number of times value is found in the collection

[\*\*.distinct\( {options : Integer} \) : Collection\*\*](#)

[\*\*.distinct\( propertyPath : Text {; options : Integer } \) : Collection\*\*](#) returns a collection containing only distinct (different) values from the original collection

[\*\*.equal\( collection2 : Collection {; option : Integer } \) : Boolean\*\*](#) compares the collection with collection2

[\*\*.every\( { startFrom : Integer ; } formula : 4D.Function { ;...param : any } \) : Boolean\*\*](#)

[\*\*.every\( { startFrom : Integer ; } methodName : Text { ;...param : any } \) : Boolean\*\*](#) returns **true** if all elements in the collection successfully passed a test implemented in the provided *formula* object or *methodName* name

[\*\*.extract\( propertyPath : Text { ; option : Integer } \) : Collection\*\*](#)

[\*\*.extract\( propertyPath : Text ; targetPath : Text { ;...propertyPathN : Text ;...targetPathN : Text } \) : Collection\*\*](#) creates and returns a new collection containing *propertyPath* values extracted from the original collection of objects

[\*\*.fill\( value : any \) : Collection\*\*](#)

.*value* . any . collection

.fill( *value* : any ; *startFrom* : Integer { ; *end* : Integer } ) : Collection fills the collection with the specified *value*, optionally from *startFrom* index to *end* index, and returns the resulting collection

.filter( *formula* : 4D.Function { ; ...*param* : any } ) : Collection

.filter( *methodName* : Text { ; ...*param* : any } ) : Collection returns a new collection containing all elements of the original collection for which the *formula* or *methodName* result is **true**

.find( { *startFrom* : Integer ; } *formula* : 4D.Function { ; ...*param* : any } ) : any

.find( { *startFrom* : Integer ; } *methodName* : Text { ; ...*param* : any } ) : any

any returns the first value in the collection for which *formula* or *methodName* result, applied on each element, returns **true**

.findIndex( { *startFrom* : Integer ; } *formula* : 4D.Function { ; ...*param* : any } ) : Integer

.findIndex( { *startFrom* : Integer ; } *methodName* : Text { ; ...*param* : any } ) : Integer returns the index, in the collection, of the first value for which *formula* or *methodName*, applied on each element, returns **true**

.first() : any returns the first element of the collection

.flat( { *depth* : Integer } ) : Collection creates a new collection with all sub-collection elements concatenated into it recursively up to the specified *depth*

.flatMap( *formula* : 4D.Function { ; ...*param* : any } ) : Collection

.flatMap( *methodName* : Text { ; ...*param* : any } ) : Collection creates a new collection based upon the result of the call of the *formula* 4D function or *methodName* method on each element of the original collection and flattened by a depth of 1

.includes( *toSearch* : expression { ; *startFrom* : Integer } ) : Boolean returns True if the *toSearch* expression is found among collection elements, otherwise False

.indexOf( *toSearch* : expression { ; *startFrom* : Integer } ) : Integer searches the *toSearch* expression among collection elements and returns the index of the first found occurrence, or -1 if it was not found

.indices( *queryString* : Text { ; ...*value* : any } ) : Collection returns indexes, in the original collection, of object collection elements that match the *queryString* search conditions

.insert( *index* : Integer ; *element* : any ) : Collection inserts *element* at the specified *index* position in the collection instance and returns the edited collection

.join( *delimiter* : Text { ; *option* : Integer } ) : Text converts all elements of the collection to strings and concatenates them using the specified *delimiter* string as separator

.last() : any returns the last element of the collection

.lastIndexOf( *toSearch* : expression { ; *startFrom* : Integer } ) : Integer searches the *toSearch* expression among collection elements and returns the index of the last occurrence

.length : Integer returns the number of elements in the collection

.map( *formula* : 4D.Function { ; ...*param* : any } ) : Collection

.map( *methodName* : Text { ; ...*param* : any } ) : Collection creates a new collection based upon the result of the call of the *formula* 4D function or *methodName* method on each element of the original collection

.max( { *propertyPath* : Text } ) : any returns the element with the highest value in the collection

.min( { *propertyPath* : Text } ) : any returns the element with the smallest value in the collection

.orderBy( ) : Collection

.orderBy( *pathStrings* : Text ) : Collection

.orderBy( *pathObjects* : Collection ) : Collection

.orderBy( *ascOrDesc* : Integer ) : Collection returns a new collection containing all elements of the collection in the specified order

.orderBvMethod( *formula* : 4D.Function { : ...*extraParam* : expression } ) :

## Collection

.orderByMethod( methodName : Text { ; ...extraParam : expression } ) :

Collection returns a new collection containing all elements of the collection in the order defined through the *formula* 4D function or *methodName* method

.pop() : any removes the last element from the collection and returns it as the function result

.push( element : any { ;...elementN } ) : Collection appends one or more *element*(s) to the end of the collection instance and returns the edited collection

.query( queryString : Text ; ...value : any ) : Collection

.query( queryString : Text ; querySettings : Object ) : Collection returns all elements of a collection of objects that match the search conditions

.reduce( formula : 4D.Function { ; initialValue : any { ; ...param : expression } } ) : any

.reduce( methodName : Text { ; initialValue : any { ; ...param : expression } } ) : any

applies the *formula* or *methodName* callback against an accumulator and each element in the collection (from left to right) to reduce it to a single value

.reduceRight( formula : 4D.Function { ; initialValue : any { ; ...param : expression } } ) : any

.reduceRight( methodName : Text { ; initialValue : any { ; ...param : expression } } ) : any applies the *formula* or *methodName* callback against an accumulator and each element in the collection (from right to left) to reduce it to a single value

.remove( index : Integer { ; howMany : Integer } ) : Collection removes one or more element(s) from the specified *index* position in the collection and returns the edited collection

.resize( size : Integer { ; defaultValue : any } ) : Collection sets the collection length to the specified new size and returns the resized collection

.reverse( ) : Collection returns a deep copy of the collection with all its elements in reverse order

.shift() : any removes the first element of the collection and returns it as the function result

.slice( startFrom : Integer { ; end : Integer } ) : Collection returns a portion of a collection into a new collection

.some( { startFrom : Integer ; } formula : 4D.Function { ; ...param : any } ) : Boolean

.some( { startFrom : Integer ; } methodName : Text { ; ...param : any } ) : Boolean returns true if at least one element in the collection successfully passed a test implemented in the provided *formula* or *methodName* code

.sort( formula : 4D.Function { ; ...extraParam : any } ) : Collection

.sort( methodName : Text { ; ...extraParam : any } ) : Collection sorts the elements of the original collection and also returns the sorted collection

.sum( { propertyPath : Text } ) : Real returns the sum for all values in the collection instance

.unshift( value : any { ;...valueN : any } ) : Collection inserts the given *value*(s) at the beginning of the collection

## **New collection**

**New collection** { ( ...value : any ) } : Collection

### **Parameter**

### **Type**

### **Description**

|       |   |                          |
|-------|---|--------------------------|
| value | Number, Text, Date, Time, Boolean, Object, Collection, Picture, Pointer | -> Collection's value(s) |
|-------|---|--------------------------|

|        |            |                   |
|--------|------------|-------------------|
| Result | Collection | <- New collection |
|--------|------------|-------------------|

## **Description**

The `New collection` command creates a new empty or prefilled collection and returns its reference.

If you do not pass any parameters, `New collection` creates an empty collection and returns its reference.

You must assign the returned reference to a 4D variable of the Collection type.

Keep in mind that `var : Collection` or `C_COLLECTION` statements declare a variable of the `Collection` type but does not create any collection.

Optionally, you can prefill the new collection by passing one or several *value(s)* as parameter(s).

Otherwise, you can add or modify elements subsequently through assignment. For example:

```
myCol[10]:="My new element"
```

If the new element index is beyond the last existing element of the collection, the collection is automatically resized and all new intermediary elements are assigned a **null** value.

You can pass any number of values of any supported type (number, text, date, picture, pointer, object, collection...). Unlike arrays, collections can mix data of different types.

You must pay attention to the following conversion issues:

- If you pass a pointer, it is kept "as is"; it is evaluated using the `JSON Stringify` command
- Dates are stored as "yyyy-mm-dd" dates or strings with the "YYYY-MM-DDTHH:mm:ss.SSSZ" format, according to the current "dates inside objects" database setting. When converting 4D dates into text prior to storing them in the collection, by default the program takes the local time zone into account. You can modify this behavior using the `Dates inside objects` selector of the `SET DATABASE PARAMETER` command.
- If you pass a time, it is stored as a number of milliseconds (Real).

## Example 1

You want to create a new empty collection and assign it to a 4D collection variable:

```
var $myCol : Collection  
$myCol:=New collection  
//$/myCol=[]
```

## Example 2

You want to create a prefilled collection:

```
var $filledColl : Collection  
$filledColl:=New collection(33;"mike";"november";->myPtr;Current date)  
//$/filledColl=[33,"mike","november","->myPtr","2017-03-  
28T22:00:00.000Z"]
```

## Example 3

You create a new collection and then add a new element:

```

var $coll : Collection
$coll:=New collection("a";"b";"c")
//$coll=["a","b","c"]
$coll[9]:="z" //add a 10th element with value "z"
$vcolSize:=$coll.length //10
//$coll=["a","b","c",null,null,null,null,null,"z"]

```

## New shared collection

- History

**New shared collection** {*( ...value : any )*} : Collection

| Parameter | Type  | Description                     |
|-----------|---|---------------------------------|
| value     | Number, Text, Date, Time, Boolean, Shared object, Shared collection | -> Shared collection's value(s) |
| Result    | Collection  | <- New shared collection        |

## Description

The `New shared collection` command creates a new empty or prefilled shared collection and returns its reference.

Adding an element to this collection using the assignment operator must be surrounded by the `Use...End use` structure, otherwise an error is generated (this is not necessary when adding elements using functions such as `push()` or `map()` because they automatically trigger an internal `Use...End use`). Reading an element without a `Use...End use` structure is, however, possible.

### ⓘ INFO

For more information on shared collections, please refer to the [Shared objects and collections](#) page.

If you do not pass any parameters, `New shared collection` creates an empty shared collection and returns its reference.

You must assign the returned reference to a 4D variable of the `Collection` type.

Keep in mind that `var : Collection` or `C_COLLECTION` statements declare a variable of the `Collection` type but do not create any collection.

Optionally, you can prefill the new shared collection by passing one or several *value(s)* as parameter(s). Otherwise, you can add or modify elements subsequently through object notation assignment (see example).

If the new element index is beyond the last existing element of the shared collection, the collection is automatically resized and all new intermediary elements are assigned a `null` value.

You can pass any number of values of the following supported types:

- number (real, longint...). Number values are always stored as reals.
- text
- boolean
- date
- time (stored as number of milliseconds - real)
- null

- shared object(\*)
- shared collection(\*)

## NOTE

Unlike standard (not shared) collections, shared collections do not support pictures, pointers, and objects or collections that are not shared.

(\*)When a shared object or collection is added to a shared collection, they share the same *locking identifier*. For more information on this point, refer to [4D Doc Center](#).

## Example

```
$mySharedCol:=New shared collection ("alpha"; "omega")
Use ($mySharedCol)
    $mySharedCol[1]:="beta"
End use
```

## .at()

- History

**.at( index : Integer ) : any**

| Parameter | Type      | Description                  |
|-----------|-----------|------------------------------|
| index     | Integer-> | Index of element to return   |
| Result    | any       | <- The element at that index |

## Description

The `.at()` function returns the item at position *index*, allowing for positive and negative integers.

This function does not modify the original collection.

Negative integers count back from the last item in the collection.

The function returns Undefined if *index* is beyond collection limits.

## Example

```
var $col : Collection
$col:=New collection(10; 20; 30; 40; 50)
$element:=$col.at(0) // 10
$element:=$col.at(1) // 20
$element:=$col.at(-1) // 50
$element:=$col.at(-2) // 40
$element:=$col.at(10) // undefined
```

## .average()

- History

**.average( {propertyPath : Text } ) : Real**

| Parameter    | Type            | Description  |
|--------------|-----------------|--|
| propertyPath | Text            | -> Object property path to be used for calculation |
| Result       | Real, Undefined | <- Arithmetic mean (average) of collection values  |

## Description

The `.average()` function returns the arithmetic mean (average) of defined values in the collection instance.

Only numerical elements are taken into account for the calculation (other element types are ignored).

If the collection contains objects, pass the *propertyPath* parameter to indicate the object property to take into account.

`.average()` returns `undefined` if:

- the collection is empty,
- the collection does not contain numerical elements,
- *propertyPath* is not found in the collection.

## Example 1

```
var $col : Collection  
$col:=New collection(10;20;"Monday";True;6)  
$vAvg:=$col.average() //12
```

## Example 2

```
var $col : Collection  
$col:=New collection  
$col.push(New object("name";"Smith";"salary";10000))  
$col.push(New object("name";"Wesson";"salary";50000))  
$col.push(New object("name";"Gross";"salary";10500))  
$vAvg:=$col.average("salary") //23500
```

## .clear()

- History

**.clear()** : Collection

| Parameter | Type          | Description                                   |
|-----------|---------------|---|
| Result    | Collection <- | Original collection with all elements removed |

## Description

The `.clear()` function removes all elements from the collection instance and returns an empty collection.

This function modifies the original collection.

## Example

```
var $col : Collection  
$col:=New collection(1;2;5)  
$col.clear()  
$vSize:=$col.length // $vSize=0
```

## .combine()

- History

## .combine( *col2* : Collection {; *index* : Integer } ) : Collection

| Parameter    | Type  | Description |
|--------------|---|-------------|
| <i>col2</i>  | Collection -> Collection to combine   |             |
| <i>index</i> | Integer -> Position to which insert elements to combine in collection<br>(default=length+1) |             |
| Result       | Collection <-Original collection containing combined element(s)                             |             |

### Description

The `.combine()` function inserts *col2* elements at the end or at the specified *index* position in the collection instance and returns the edited collection. Unlike the `.insert()` function, `.combine()` adds each value of *col2* in the original collection, and not as a single collection element.

This function modifies the original collection.

By default, *col2* elements are added at the end of the orginal collection. You can pass in *index* the position where you want the *col2* elements to be inserted in the collection.

**Warning:** Keep in mind that collection elements are numbered from 0.

- If *index* > the length of the collection, the actual starting *index* will be set to the length of the collection.
- If *index* < 0, it is recalculated as *index*:=*index*+*length* (it is considered as the offset from the end of the collection).
- If the calculated value is negative, *index* is set to 0.

### Example

```
var $c; $fruits : Collection
$c:=New collection(1;2;3;4;5;6)
$fruits:=New collection("Orange";"Banana";"Apple";"Grape")
$c.combine($fruits;3)
//[1,2,3,"Orange","Banana","Apple","Grape",4,5,6]
```

## .concat()

- History

## .concat( *value* : any { ;...*valueN* } ) : Collection

| Parameter    | Type   | Description  |
|--------------|--|--|
| <i>value</i> | Number, Text, Object,<br>Collection, Date, Time,<br>Boolean, Picture | Value(s) to concatenate. If <i>value</i> is a<br>->collection, all collection elements are added<br>to the original collection |
| Result       | Collection   | <- New collection with value(s) added to the<br>original collection  |

### Description

The `.concat()` function returns a new collection containing the elements of the original collection with all elements of the *value* parameter added to the end.

This function does not modify the original collection.

If *value* is a collection, all its elements are added as new elements at the end of the

original collection. If *value* is not a collection, it is added itself as a new element.

## Example

```
var $c : Collection
$c:=New collection(1;2;3;4;5)
$fruits:=New collection("Orange";"Banana";"Apple";"Grape")
$fruits.push(New object("Intruder";"Tomato"))
$c2:=$c.concat($fruits) // [1,2,3,4,5,"Orange", "Banana", "Apple", "Grape",
 {"Intruder":"Tomato"}]
$c2:=$c.concat(6;7;8) // [1,2,3,4,5,6,7,8]
```

## .copy()

- History

**.copy()** : Collection  
**.copy( option : Integer )** : Collection  
**.copy( option : Integer ; groupWithCol : Collection )** : Collection  
**.copy( option : Integer ; groupWithObj : Object )** : Collection

| Parameter    | Type       | Description  |
|--------------|------------|--|
| option       | Integer    | -><br>ck resolve pointers: resolve pointers before copying,<br>ck shared: return a shared collection |
| groupWithCol | Collection | -> Shared collection to be grouped with the resulting collection                                     |
| groupWithObj | Object     | -> Shared object to be grouped with the resulting collection   |
| Result       | Collection | <- Deep copy of the original collection  |

## Description

The `.copy()` function returns a deep copy of the collection instance. **Deep copy** means that objects or collections within the original collection are duplicated and do not share any reference with the returned collection.

This function does not modify the original collection.

If passed, the *option* parameter can contain one of the following constants (or both):

| option    | Description  |
|-----------|--|
| ck        | If the original collection contains pointer type values, by default the copy also contains the pointers. However, you can resolve pointers when copying by passing the <code>ck resolve pointers</code> constant. In this case, each pointer present in the collection is evaluated when copying and its dereferenced value is used.                 |
| ck shared | By default, <code>copy()</code> returns a regular (not shared) collection, even if the command is applied to a shared collection. Pass the <code>ck shared</code> constant to create a shared collection. In this case, you can use the <i>groupWith</i> parameter to associate the shared collection with another collection or object (see below). |

The *groupWithCol* or *groupWithObj* parameters allow you to designate a collection or an object with which the resulting collection should be associated.

### ⓘ NOTE

Datastore, dataclass, and entity objects are not copiable. If `.copy()` is called with

them, `Null` values are returned.

## Example 1

We want to copy the `$lastnames` regular (non shared) collection into the `$sharedObject` shared object. To do this, we must create a shared copy of the collection (`$sharedLastnames`).

```
var $sharedObject : Object
var $lastnames;$sharedLastnames : Collection
var $text : Text

$sharedObject:=New shared object

$text:=Document to text(Get 4D folder(�urrent resources
folder)+"lastnames.txt")
$lastnames:=JSON Parse($text) // $lastnames is a regular collection

$sharedLastnames:=$lastnames.copy(ck shared) // $sharedLastnames is a
shared collection

// Now we can put $sharedLastnames into $sharedObject
Use($sharedObject)
    $sharedObject.lastnames:=$sharedLastnames
End use
```

## Example 2

We want to combine `$sharedColl1` and `$sharedColl2`. Since they belong to different shared groups, a direct combination would result in an error. Therefore, we must make a shared copy of `$sharedColl1` and designate `$sharedColl2` as a shared group for the copy.

```
var $sharedColl1;$sharedColl2;$copyColl : Collection

$sharedColl1:=New shared collection(New shared
object("lastname";"Smith"))
$sharedColl2:=New shared collection(New shared
object("lastname";"Brown"))

// $copyColl belongs to the same shared group as $sharedColl2
$copyColl:=$sharedColl1.copy(ck shared;$sharedColl2)
Use($sharedColl2)
    $sharedColl2.combine($copyColl)
End use
```

## Example 3

We have a regular collection (`$lastnames`) and we want to put it in the **Storage** of the application. To do this, we must create a shared copy beforehand (`$sharedLastnames`).

```

var $lastnames;$sharedLastnames : Collection
var $text : Text

$text:=Document to text(Get 4D folder(�urrent resources
folder)+"lastnames.txt")
$lastnames:=JSON Parse($text) // $lastnames is a regular collection

$sharedLastnames:=$lastnames.copy(ck shared) // shared copy

Use(Storage)
    Storage.lastnames:=$sharedLastnames
End use

```

## Example 4

This example illustrates the use of the `ck resolve pointers` option:

```

var $col : Collection
var $p : Pointer
$p:=->$what

$col:=New collection
$col.push(New object("alpha";"Hello";"num";1))
$col.push(New object("beta";"You";"what";$p))

$col2:=$col.copy()
$col2[1].beta:="World!"
ALERT($col[0].alpha+" "+$col2[1].beta) //displays "Hello World!"

$what:="You!"
$col3:=$col2.copy(ck resolve pointers)
ALERT($col3[0].alpha+" "+$col3[1].what) //displays "Hello You!"

```

## .count()

- History

**.count( { *propertyPath* : Text } ) : Real**

| Parameter    | Type    | Description                                     |
|--------------|---------|---|
| propertyPath | Text -> | Object property path to be used for calculation |
| Result       | Real <- | Number of elements in the collection            |

### Description

The `.count()` function returns the number of non-null elements in the collection.

If the collection contains objects, you can pass the *propertyPath* parameter. In this case, only elements that contain the *propertyPath* are taken into account.

## Example

```

var $col : Collection
var $count1;$count2 : Real
$col:=New collection(20;30;Null;40)
$col.push(New object("name";"Smith";"salary";10000))
$col.push(New object("name";"Wesson";"salary";50000))
$col.push(New object("name";"Gross";"salary";10500))
$col.push(New object("lastName";"Henry";"salary";12000))
$count1:=$col.count() // $count1=7
$count2:=$col.count("name") // $count2=3

```

## .countValues()

- History

**.countValues( value : any {; propertyPath : Text } ) : Real**

| Parameter    | Type  | Description  |
|--------------|---|--|
| value        | Text, Number, Boolean, Date, Object, Collection | -> Value to count                                  |
| propertyPath | Text  | -> Object property path to be used for calculation |
| Result       | Real  | <- Number of occurrences of the value              |

### Description

The `.countValues()` function returns the number of times *value* is found in the collection.

You can pass in *value*:

- a scalar value (text, number, boolean, date),
- an object or a collection reference.

For an element to be found, the type of *value* must be equivalent to the type of the element; the method uses the equality operator.

The optional *propertyPath* parameter allows you to count values inside a collection of objects: pass in *propertyPath* the path of the property whose values you want to count.

This function does not modify the original collection.

### Example 1

```

var $col : Collection
var $vCount : Integer
$col:=New collection(1;2;5;5;5;3;6;4)
$vCount:=$col.countValues(5) // $vCount=3

```

### Example 2

```

var $col : Collection
var $vCount : Integer
$col:=New collection
$col.push(New object("name";"Smith";"age";5))
$col.push(New object("name";"Wesson";"age";2))
$col.push(New object("name";"Jones";"age";3))
$col.push(New object("name";"Henry";"age";4))
$col.push(New object("name";"Gross";"age";5))
$vCount:=$col.countValues(5;"age") // $vCount=2

```

### Example 3

```

var $numbers; $letters : Collection
var $vCount : Integer

$letters:=New collection("a";"b";"c")
$numbers:=New collection(1;2;$letters;3;4;5)

$vCount:=$numbers.countValues($letters) // $vCount=1

```

## .distinct()

- History

**.distinct( {options : Integer} ) : Collection**  
**.distinct( propertyPath : Text {; options : Integer } ) : Collection**

| Parameter    | Type       | Description  |
|--------------|------------|--|
| propertyPath | Text       | -> Path of attribute whose distinct values you want to get |
| options      | Integer    | -> ck diacritical, ck count values                         |
| Result       | Collection | <- New collection with only distinct values                |

### Description

The `.distinct()` function returns a collection containing only distinct (different) values from the original collection.

This function does not modify the original collection.

The returned collection is automatically sorted. **Null** values are not returned.

If the collection contains objects, you can pass the *propertyPath* parameter to indicate the object property whose distinct values you want to get.

In the *options* parameter, you can pass one or a combination of the following constants:

| Constant        | Value | Comment   |
|-----------------|-------|---|
| ck diacritical  | 8     | Evaluation is case sensitive and differentiates accented characters. By default if omitted, a non-diacritical evaluation is performed   |
| ck count values | 32    | Return the count of elements for every distinct value. When this option is passed, <code>.distinct()</code> returns a collection of objects containing a pair of {"value": *value*, "count": *count*} attributes. |

### Examples

```

var $c; $c2; $c3 : Collection
$c:=New collection
$c.push("a";"b";"c";"A";"B";"c";"b";"b")
$c.push(New object("size";1))
$c.push(New object("size";3))
$c.push(New object("size";1))
$c2:=$c.distinct() // $c2=[{"a", "b", "c", {"size":1}, {"size":3}, {"size":1}]
$c2:=$c.distinct(ck diacritical) // $c2=[{"a", "A", "b", "B", "c", {"size":1}, {"size":3}, {"size":1}]
$c2:=$c.distinct("size") // $c2=[1, 3]
$c3:=$c.distinct("size";ck count values) // $c3=[{"value":1, "count":2}, {"value":3, "count":1}]

```

## .equal()

- History

**.equal( collection2 : Collection {; option : Integer } ) : Boolean**

| Parameter   | Type                      | Description   |
|-------------|---------------------------|---|
| collection2 | Collection -> Collection  | to compare  |
| option      | Integer -> ck diacritical | : diacritical evaluation ("A" # "a" for example)      |
| Result      | Boolean                   | <- True if collections are identical, false otherwise |

### Description

The `.equal()` function compares the collection with collection2 and returns **true** if they are identical (deep comparison).

By default, a non-diacritical evaluation is performed. If you want the evaluation to be case sensitive or to differentiate accented characters, pass the `ck diacritical` constant in the option parameter.

Elements with **Null** values are not equal to Undefined elements.

### Example

```

var $c; $c2 : Collection
var $b : Boolean

$c:=New collection(New object("a";1;"b";"orange");2;3)
$c2:=New collection(New object("a";1;"b";"orange");2;3;4)
$b:=$c.equal($c2) // false

$c:=New collection(New object("1";"a";"b";"orange");2;3)
$c2:=New collection(New object("a";1;"b";"orange");2;3)
$b:=$c.equal($c2) // false

$c:=New collection(New object("a";1;"b";"orange");2;3)
$c2:=New collection(New object("a";1;"b";"ORange");2;3)
$b:=$c.equal($c2) // true

$c:=New collection(New object("a";1;"b";"orange");2;3)
$c2:=New collection(New object("a";1;"b";"ORange");2;3)
$b:=$c.equal($c2);ck diacritical) //false

```

## .every()

- History

.every( { startFrom : Integer ; } formula : 4D.Function { ;...param : any } ) : Boolean  
.every( { startFrom : Integer ; } methodName : Text { ;...param : any } ) : Boolean

| Parameter  | Type        | Description  |
|------------|-------------|--|
| startFrom  | Integer     | -> Index to start the test at                                  |
| formula    | 4D.Function | -> Formula object  |
| methodName | Text        | -> Name of a method  |
| param      | Mixed       | -> Parameter(s) to pass to <i>formula</i> or <i>methodName</i> |
| Result     | Boolean     | <- True if all elements successfully passed the test           |

## Description

The `.every()` function returns **true** if all elements in the collection successfully passed a test implemented in the provided *formula* object or *methodName* name.

You designate the callback to be executed to evaluate collection elements using either:

- *formula* (recommended syntax), a [Formula object](#) that can encapsulate any executable expressions, including functions and project methods;
- or *methodName*, the name of a project method (text).

The callback is called with the parameter(s) passed in *param* (optional). The callback can perform any test, with or without the parameter(s) and must return **true** for every element fulfilling the test. It receives an `Object` in first parameter (\$1).

The callback receives the following parameters:

- in `$1.value`: element value to be evaluated
- in `$2`: param
- in `$N...`: paramN...

It can set the following parameter(s):

- (mandatory if you used a method) `$1.result` (Boolean): **true** if the element value evaluation is successful, **false** otherwise.
- `$1.stop` (Boolean, optional): **true** to stop the method callback. The returned value is the last calculated.

In all cases, at the point when the `.every()` function encounters the first collection element evaluated to **false**, it stops calling the callback and returns **false**.

By default, `.every()` tests the whole collection. Optionally, you can pass in *startFrom* the index of the element from which to start the test.

- If *startFrom*  $\geq$  the collection's length, **false** is returned, which means the collection is not tested.
- If *startFrom*  $< 0$ , it is considered as the offset from the end of the collection ( $startFrom := startFrom + length$ ).
- If *startFrom* = 0, the whole collection is searched (default).

## Example 1

```

var $c : Collection
var $b : Boolean
var $f : 4D.Function

$f:=Formula($1.value>0)
$c:=New collection
$c.push(5;3;1;4;6;2)
$b:=$c.every($f) //returns true
$c.push(-1)
$b:=$c.every($f) //returns false

```

## Example 2

This example tests that all elements of a collection are of the real type:

```

var $c : Collection
var $b : Boolean
var $f : 4D.Function

$f:=Formula(Value type($1.value)==$2
$c:=New collection
$c.push(5;3;1;4;6;2)
$b:=$c.every($f;Is real) //b=true
$c:=$c.push(New object("name";"Cleveland";"zc";35049))
$c:=$c.push(New object("name";"Blountsville";"zc";35031))
$b:=$c.every($f;Is real) //b=false

```

## .extract()

- History

**.extract(*propertyPath* : Text { ; *option* : Integer } ) : Collection**  
**.extract(*propertyPath* : Text ; *targetPath* : Text { ;...*propertyPathN* : Text ;...*targetPathN* : Text } ) : Collection**

| Parameter           | Type       | Description   |
|---------------------|------------|---|
| <i>propertyPath</i> | Text       | -> Object property path whose values must be extracted to the new collection                      |
| <i>targetpath</i>   | Text       | -> Target property path or property name<br>ck keep null: include null properties in the returned |
| <i>option</i>       | Integer    | -> collection (ignored by default). Parameter ignored if <i>targetPath</i> passed.                |
| Result              | Collection | <- New collection containing extracted values   |

## Description

The `.extract()` function creates and returns a new collection containing *propertyPath* values extracted from the original collection of objects.

This function does not modify the original collection.

The contents of the returned collection depends on the *targetPath* parameter:

- If the *targetPath* parameter is omitted, `.extract()` populates the new collection with the *propertyPath* values of the original collection.

By default, elements for which *propertyPath* is null or undefined are ignored in the resulting collection. You can pass the `ck keep null` constant in the *option* parameter to include these values as null elements in the returned collection.

- If one or more *targetPath* parameter(s) are passed, `.extract()` populates the new collection with the *propertyPath* properties and each element of the new collection is an object with *targetPath* properties filled with the corresponding *propertyPath* properties. Null values are kept (*option* parameter is ignored with this syntax).

## Example 1

```
var $c : Collection
$c:=New collection
$c.push(New object("name";"Cleveland"))
$c.push(New object("zip";5321))
$c.push(New object("name";"Blountsville"))
$c.push(42)
$c2:=$c.extract("name") // $c2=[Cleveland,Blountsville]
$c2:=$c.extract("name";ck keep null) // $c2=
[Cleveland,null,Blountsville,null]
```

## Example 2

```
var $c : Collection
$c:=New collection
$c.push(New object("zc";35060))
$c.push(New object("name";Null;"zc";35049))
$c.push(New object("name";"Cleveland";"zc";35049))
$c.push(New object("name";"Blountsville";"zc";35031))
$c.push(New object("name";"Adger";"zc";35006))
$c.push(New object("name";"Clanton";"zc";35046))
$c.push(New object("name";"Clanton";"zc";35045))
$c2:=$c.extract("name";"City") // $c2=[{City:null},{City:Cleveland},
{City:Blountsville},{City:Adger},{City:Clanton},{City:Clanton}]
$c2:=$c.extract("name";"City";"zc";"Zip") // $c2=[{Zip:35060},
{City:null,Zip:35049},{City:Cleveland,Zip:35049},
{City:Blountsville,Zip:35031},{City:Adger,Zip:35006},
{City:Clanton,Zip:35046},{City:Clanton,Zip:35045}]
```

## .fill()

- History

`.fill( value : any ) : Collection`

`.fill( value : any ; startFrom : Integer { ; end : Integer } ) : Collection`

| Parameter | Type  | Description                               |
|-----------|---|---|
| value     | number, Text, Collection, Object, Date, Boolean | -> Filling value                          |
| startFrom | Integer   | -> Start index (included)                 |
| end       | Integer   | -> End index (not included)               |
| Result    | collection                                      | <- Original collection with filled values |

## Description

The `.fill()` function fills the collection with the specified *value*, optionally from *startFrom* index to *end* index, and returns the resulting collection.

This function modifies the original collection.

- If the *startFrom* parameter is omitted, *value* is set to all collection elements (*startFrom=0*).

- If the *startFrom* parameter is passed and *end* omitted, *value* is set to collection elements starting at *startFrom* to the last element of the collection (*end*=length).
- If both the *startFrom* parameter and *end* are passed, *value* is set to collection elements starting at *startFrom* to the element *end*.

In case of inconsistency, the following rules apply:

- If *startFrom* < 0, it is recalculated as *startFrom*:=*startFrom*+*length* (it is considered as the offset from the end of the collection). If the calculated value is negative, *startFrom* is set to 0.
- If *end* < 0 , it is recalculated as *end*:=*end*+*length*.
- If *end* < *startFrom* (passed or calculated values), the method does nothing.

## Example

```
var $c : Collection
$c:=New collection(1;2;3;"Lemon";Null;"";4;5)
$c.fill("2") // $c:=[2,2,2,2,2,2,2]
$c.fill("Hello";5) // $c=[2,2,2,2,2>Hello,Hello,Hello]
$c.fill(0;1;5) // $c=[2,0,0,0,0>Hello,Hello,Hello]
$c.fill("world";1;-5) // -5+8=3 -> $c=
[2,"world","world",0,0>Hello,Hello,Hello]
```

## .filter()

- History

**.filter( formula : 4D.Function { ; ...param : any } ) : Collection**  
**.filter( methodName : Text { ; ...param : any } ) : Collection**

| Parameter  | Type                        | Description  |
|------------|-----------------------------|--|
| formula    | 4D.Function->Formula object |  |
| methodName | Text                        | -> Name of a method  |
| param      | any                         | -> Parameter(s) to pass to <i>formula</i> or <i>methodName</i> |
| Result     | Collection                  | <- New collection containing filtered elements (shallow copy)  |

## Description

The `.filter()` function returns a new collection containing all elements of the original collection for which the *formula* or *methodName* result is **true**. This function returns a **shallow copy**, which means that objects or collections in both collections share the same reference. If the original collection is a shared collection, the returned collection is also a shared collection.

This function does not modify the original collection.

You designate the callback to be executed to filter collection elements using either:

- *formula* (recommended syntax), a [Formula object](#) that can encapsulate any executable expressions, including functions and project methods;
- or *methodName*, the name of a project method (text).

The callback is called with the parameter(s) passed in *param* (optional) and an object in first parameter (\$1). The callback can perform any test, with or without the parameter(s) and must return **true** for each element fulfilling the condition and thus, to push to the new collection.

The callback receives the following parameters:

- in `$1.value`: element value to be evaluated
- in `$2`: param
- in `$N...`: paramN...

It can set the following parameter(s):

- `$1.result` (Boolean): **true** if the element value matches the filter condition and must be kept, **false** otherwise.
- `$1.stop` (Boolean, optional): **true** to stop the method callback. The returned value is the last calculated.

### NOTE

When using `methodName` as callback, and if the method does not return any value, `.filter()` will look at the property `$1.result` that you must set to **true** for each element fulfilling the condition.

### Example 1

You want to get the collection of text elements whose length is smaller than 6:

```
var $col;$colNew : Collection
$col:=New collection("hello";"world";"red horse";66;"tim";"san
jose";"miami")
$colNew:=$col.filter(Formula((Value type($1.value)=Is text) &&
(Length($1.value)<$2)); 6)
// $colNew=["hello","world","tim","miami"]
```

### Example 2

You want to filter elements according to their value type:

```
var $c;$c2;$c3 : Collection
var $f : 4D.Function

$f:=Formula(OB Get type($1;"value")=$2)
$c:=New collection(5;3;1;4;6;2)
$c.push(New object("name";"Cleveland";"zc";35049))
$c.push(New object("name";"Blountsville";"zc";35031))
$c2:=$c.filter($f;Is real) // $c2=[5,3,1,4,6,2]
$c3:=$c.filter($f;Is object)
// $c3=[{name:Cleveland,zc:35049},{name:Blountsville,zc:35031}]
```

## .find()

- History

**.find( { startFrom : Integer ; } formula : 4D.Function { ; ...param : any } ) : any**  
**.find( { startFrom : Integer ; } methodName : Text { ; ...param : any } ) : any**

| Parameter  | Type        | Description  |
|------------|-------------|--|
| startFrom  | Integer     | ->Index to start the search at                                 |
| formula    | 4D.Function | ->Formula object   |
| methodName | Text        | ->Name of a method   |
| param      | any         | -> Parameter(s) to pass to <i>formula</i> or <i>methodName</i> |
| Result     | any         | <-First value found, or Undefined if not found                 |

### Description

The `.find()` function returns the first value in the collection for which *formula* or *methodName* result, applied on each element, returns **true**.

This function does not modify the original collection.

You designate the callback to be executed to evaluate collection elements using either:

- *formula* (recommended syntax), a [Formula object](#) that can encapsulate any executable expressions, including functions and project methods;
- or *methodName*, the name of a project method (text).

The callback is called with the parameter(s) passed in *param* (optional). The callback can perform any test, with or without the parameter(s) and must return **true** for the first element fulfilling the condition. It receives an `Object` in first parameter (\$1).

The callback receives the following parameters:

- in `$1.value`: element value to be evaluated
- in `$2`: param
- in `$N...`: paramN...

It can set the following parameter(s):

- (mandatory if you used a method) `$1.result` (Boolean): **true** if the element value matches the search condition, **false** otherwise.
- `$1.stop` (Boolean, optional): **true** to stop the method callback. The returned value is the last calculated.

By default, `.find()` searches in the whole collection. Optionally, you can pass in *startFrom* the index of element from which to start the search.

- If *startFrom*  $\geq$  the collection's length, -1 is returned, which means the collection is not searched.
- If *startFrom*  $< 0$ , it is considered as the offset from the end of the collection (*startFrom*:=*startFrom*+*length*). **Note:** Even if *startFrom* is negative, the collection is still searched from left to right.
- If *startFrom* = 0, the whole collection is searched (default).

## Example 1

You want to get the first text element with a length smaller than 5:

```
var $col : Collection
$col:=New collection("hello";"world";4;"red horse";"tim";"san jose")
$value:=$col.find(Formula((Value type($1.value)=Is text) &&
(Length($1.value)<$2)); 5) //$/value="tim"
```

## Example 2

You want to find a city name within a collection:

```

var $c : Collection
var $c2 : Object
$c:=New collection
$c.push(New object("name"; "Cleveland"; "zc"; 35049))
$c.push(New object("name"; "Blountsville"; "zc"; 35031))
$c.push(New object("name"; "Adger"; "zc"; 35006))
$c.push(New object("name"; "Clanton"; "zc"; 35046))
$c.push(New object("name"; "Clanton"; "zc"; 35045))

$c2:=$c.find(Formula($1.value.name=$2); "Clanton") // $c2=
{name:Clanton, zc:35046}

```

## .findIndex()

- History

**.findIndex( { startFrom : Integer ; } formula : 4D.Function { ; ...param : any } ) :**  
**Integer**

**.findIndex( { startFrom : Integer ; } methodName : Text { ; ...param : any } ) :**  
**Integer**

| Parameter  | Type        | Description   |
|------------|-------------|---|
| startFrom  | Integer     | -> Index to start the search at                                   |
| formula    | 4D.Function | -> Formula object   |
| methodName | Text        | -> Name of a method   |
| param      | any         | -> Parameter(s) to pass to <i>formula</i> or<br><i>methodName</i> |
| Result     | Integer     | <- Index of first value found, or -1 if not found                 |

### Description

The `.findIndex()` function returns the index, in the collection, of the first value for which *formula* or *methodName*, applied on each element, returns **true**.

This function does not modify the original collection.

You designate the callback to be executed to evaluate collection elements using either:

- *formula* (recommended syntax), a [Formula object](#) that can encapsulate any executable expressions, including functions and project methods;
- *methodName*, the name of a project method (text).

The callback is called with the parameter(s) passed in *param* (optional). The callback can perform any test, with or without the parameter(s) and must return **true** for the first element fulfilling the condition. It receives an `Object` in first parameter (\$1).

The callback receives the following parameters:

- in `$1.value`: element value to be evaluated
- in `$2`: param
- in `$N...`: paramN...

It can set the following parameter(s):

- (mandatory if you used a method) `$1.result` (Boolean): **true** if the element value matches the search condition, **false** otherwise.
- `$1.stop` (Boolean, optional): **true** to stop the method callback. The returned value is the last calculated.

By default, `.findIndex()` searches in the whole collection. Optionally, you can pass in `startFrom` the index of element from which to start the search.

- If `startFrom >=` the collection's length, -1 is returned, which means the collection is not searched.
- If `startFrom < 0`, it is considered as the offset from the end of the collection (`startFrom:=startFrom+length`). **Note:** Even if `startFrom` is negative, the collection is still searched from left to right.
- If `startFrom = 0`, the whole collection is searched (default).

## Example

You want to find the position of the first city name within a collection:

```
var $c : Collection
var $val2;$val3 : Integer
$c:=New collection
$c.push(New object("name";"Cleveland";"zc";35049))
$c.push(New object("name";"Blountsville";"zc";35031))
$c.push(New object("name";"Adger";"zc";35006))
$c.push(New object("name";"Clanton";"zc";35046))
$c.push(New object("name";"Clanton";"zc";35045))
$val2:=$c.findIndex(Formula($1.value.name=$2);"Clanton") // $val2=3
$val3:=$c.findIndex($val2+1;Formula($1.value.name=$2);"Clanton")
//$val3=4
```

## .first()

- History

`.first()` : any

### Parameter Type Description

Result any <-First element of collection

## Description

The `.first()` function returns the first element of the collection.

This function does not modify the original collection.

The function returns Undefined if the collection is empty.

## Example

```
var $col; $emptyCol : Collection
var $first : Variant
$col:=New collection(10; 20; 30; "hello"; 50)
$first:=$col.first() // 10

$emptyCol:=New collection() //empty
// $first:=$emptyCol[0] //would return error
$first:=$emptyCol.first() // returns Undefined
```

## .flat()

- History

`.flat( { depth : Integer } )` : Collection

| Parameter | Type       | Description   |
|-----------|------------|---|
| depth     | Integer    | -> How deep a nested collection structure should be flattened.<br>Default=1 |
| Result    | Collection | <- Flattened collection   |

## Description

The `.flat()` function creates a new collection with all sub-collection elements concatenated into it recursively up to the specified *depth*.

By default, if the *depth* parameter is omitted, only the first level of the nested collection structure will be flattened.

This function does not modify the original collection.

## Example

```
$col:=New collection(1; 2; New collection(3; 4))
$col.flat()
// [1, 2, 3, 4]

$col:=New collection(1; 2; New collection(3; 4; New collection(5; 6)))
$col.flat()
// [1, 2, 3, 4, [5, 6]]

$col:=New collection(1; 2; New collection(3; 4; New collection(5; 6)))
$col.flat(2)
// [1, 2, 3, 4, 5, 6]

$col:=New collection(1; 2; New collection(3; 4; 5; 6; New collection(7;
8; New collection(9; 10))))
$col.flat(MAXLONG)
// [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

## .flatMap()

- History

`.flatMap( formula : 4D.Function { ; ...param : any } )` : Collection  
`.flatMap( methodName : Text { ; ...param : any } )` : Collection

| Parameter  | Type                        | Description   |
|------------|-----------------------------|---|
| formula    | 4D.Function->Formula object |   |
| methodName | Text                        | -> Name of a method   |
| param      | any                         | -> Parameter(s) to pass to <i>formula</i> or <i>methodName</i>    |
| Result     | Collection                  | <- Collection of transformed values and flattened by a depth of 1 |

## Description

The `.flatMap()` function creates a new collection based upon the result of the call of the *formula* 4D function or *methodName* method on each element of the original collection and flattened by a depth of 1. Optionally, you can pass parameters to *formula* or *methodName* using the *param* parameter(s).

This function is identical to a `map()` call followed by a `flat()` call of depth 1.

This function does not modify the original collection.

You designate the callback to be executed to evaluate collection elements using either:

- *formula* (recommended syntax), a [Formula object](#) that can encapsulate any executable expressions, including functions and project methods;
- or *methodName*, the name of a project method (text).

The callback is called with the parameter(s) passed in *param* (optional). The callback can perform any operation, with or without the parameter(s) and must return new transformed value to add to the resulting collection. It receives an `Object` in first parameter (\$1).

The callback receives the following parameters:

- in `$1.value`: element value to be evaluated
- in `$2`: param
- in `$N...`: paramN...

It can set the following parameter(s):

- (mandatory if you used a method) `$1.result` (any type): new transformed value to add to the resulting collection
- `$1.stop` (Boolean, optional): `true` to stop the method callback. The returned value is the last calculated.

## Example 1

```
var $col ; $result : Collection
$col:=New collection(1; 2; 3; 4)

$result:=$col.map(Formula(New collection($1.value*2)))
// [[2],[4],[6],[8]]

$result:=$col.flatMap(Formula(New collection($1.value*2)))
// [2,4,6,8]
```

## Example 2

```
var $col; $result : Collection
$col:=New collection("Hello how"; ""; "are you ?")

$result:=$col.map(Formula(Split string($1.value; " ")))
// [["Hello", "how"], [], ["are", "you", "?"]]

$result:=$col.flatMap(Formula(Split string($1.value; " ")))
// ["Hello", "how", "are", "you", "?"]
```

## Example 3

You want to compute the percentage of each value in the collection to the total:

```
var $c; $c2 : Collection
var $f : 4D.Function
$c:=New collection(1; 4; 9; 10; 20)
$f:=Formula(New collection($1.value;Round(($1.value/$2)*100; 2)))
$c2:=$c.flatMap($f; $c.sum())
// $c2=[1, 2.27, 4, 9.09, 9, 20.45, 10, 22.73, 20, 45.45]
```

## .includes()

- History

**.includes( *toSearch* : expression { ; *startFrom* : Integer } ) : Boolean**

| Parameter        | Type   | Description   |
|------------------|--|---|
| <i>toSearch</i>  | expression -> Expression to search in the collection |   |
| <i>startFrom</i> | Integer  | -> Index to start the search at                       |
| Result           | Boolean  | <- True if <i>toSearch</i> is found in the collection |

## Description

The `.includes()` function returns True if the *toSearch* expression is found among collection elements, otherwise False.

This function does not modify the original collection.

In *toSearch*, pass the expression to find in the collection. You can pass:

- a scalar value (text, number, boolean, date),
- the null value,
- an object or a collection reference.

*toSearch* must match exactly the element to find (the same rules as for the equality operator of the data type are applied).

Optionally, you can pass the index of collection from which to start the search in *startFrom*.

- If *startFrom*  $\geq$  collection's length, False is returned, which means the collection is not searched.
- If *startFrom*  $< 0$ , it is considered as the offset from the end of the collection (*startFrom* := *startFrom* + length). Note that even if *startFrom* is negative, the collection is still searched from left to right.
- If *startFrom* = 0, the whole collection is searched (default).

## Example

```
var $col : Collection
var $in : Boolean
var $obj : Object
$obj:=New object("value"; 10)
$col:=New collection(1;2;"Henry";5;3;"Albert";6;4;"Alan";5;$obj)
$in:=$col.includes(3) //True
$in:=$col.includes(5;6) //True
$in:=$col.includes("al@") //True
$in:=$col.includes("Hello") //False
$in:=$col.includes($obj) //True
$in:=$col.includes(New object("value"; 10)) //False
```

## .indexOf()

- History

**.indexOf( *toSearch* : expression { ; *startFrom* : Integer } ) : Integer**

| Parameter        | Type   | Description  |
|------------------|--|--|
| <i>toSearch</i>  | expression -> Expression to search in the collection |  |
| <i>startFrom</i> | Integer  | -> Index to start the search at  |
| Result           | Integer  | <- Index of the first occurrence of <i>toSearch</i> in the collection, -1 if not found |

## Description

The `.indexOf()` function searches the `toSearch` expression among collection elements and returns the index of the first found occurrence, or -1 if it was not found.

This function does not modify the original collection.

In `toSearch`, pass the expression to find in the collection. You can pass:

- a scalar value (text, number, boolean, date),
- the null value,
- an object or a collection reference.

`toSearch` must match exactly the element to find (the same rules as for the equality operator of the data type are applied).

Optionally, you can pass the index of collection from which to start the search in `startFrom`.

- If `startFrom >=` the collection's length, -1 is returned, which means the collection is not searched.
- If `startFrom < 0`, it is considered as the offset from the end of the collection (`startFrom:=startFrom+length`). **Note:** Even if `startFrom` is negative, the collection is still searched from left to right.
- If `startFrom = 0`, the whole collection is searched (default).

## Example

```
var $col : Collection
var $i : Integer
$col:=New collection(1;2;"Henry";5;3;"Albert";6;4;"Alan";5)
$i:=$col.indexOf(3) // $i=4
$i:=$col.indexOf(5;5) // $i=9
$i:=$col.indexOf("al@") // $i=5
$i:=$col.indexOf("Hello") // $i=-1
```

## .indices()

- History

`.indices( queryString : Text { ; ...value : any } ) : Collection`

| Parameter   | Type          | Description  |
|-------------|---------------|--|
| queryString | Text          | -> Search criteria                                       |
| value       | any           | -> Value(s) to compare when using placeholder(s)         |
| Result      | Collection <- | Element index(es) matching queryString in the collection |

## Description

The `.indices()` function works exactly the same as the [`.query\(\)`](#) function but returns indexes, in the original collection, of object collection elements that match the `queryString` search conditions, and not elements themselves. Indexes are returned in ascending order.

This function does not modify the original collection.

The `queryString` parameter uses the following syntax:

```
propertyPath comparator value {logicalOperator propertyPath comparator  
value}
```

For a detailed description of the *queryString* and *value* parameters, please refer to the `dataClass.query()` function.

## Example

```
var $c; $icoll : Collection  
$c:=New collection  
$c.push(New object("name";"Cleveland";"zc";35049))  
$c.push(New object("name";"Blountsville";"zc";35031))  
  
$c.push(New object("name";"Adger";"zc";35006))  
$c.push(New object("name";"Clanton";"zc";35046))  
$c.push(New object("name";"Clanton";"zc";35045))  
$icoll:=$c.indices("name = :1";"Cleveland") // $icoll=[0]  
$icoll:=$c.indices("zc > 35040") // $icoll=[0,3,4]
```

## .insert()

- History

**.insert( index : Integer ; element : any ) : Collection**

| Parameter | Type       | Description  |
|-----------|------------|--|
| index     | Integer    | -> Where to insert the element                     |
| element   | any        | -> Element to insert in the collection             |
| Result    | Collection | <- Original collection containing inserted element |

## Description

The `.insert()` function inserts *element* at the specified *index* position in the collection instance and returns the edited collection.

This function modifies the original collection.

In *index*, pass the position where you want the element to be inserted in the collection.

**Warning:** Keep in mind that collection elements are numbered from 0.

- If *index* > the length of the collection, actual starting index will be set to the length of the collection.
- If *index* < 0, it is recalculated as *index* := *index* + *length* (it is considered as the offset from the end of the collection).
- If the calculated value is negative, index is set to 0.

Any type of element accepted by a collection can be inserted, even another collection.

## Example

```
var $col : Collection  
$col:=New collection("a";"b";"c";"d") // $col=["a", "b", "c", "d"]  
$col.insert(2;"X") // $col=["a", "b", "X", "c", "d"]  
$col.insert(-2;"Y") // $col=["a", "b", "X", "Y", "c", "d"]  
$col.insert(-10;"Hi") // $col=["Hi", "a", "b", "X", "Y", "c", "d"]
```

## .join()

- History

**.join( delimiter : Text { ; option : Integer } ) : Text**

| Parameter | Type    | Description   |
|-----------|---------|---|
| delimiter | Text    | -> Separator to use between elements  |
| option    | Integer | -> <code>ck ignore null or empty</code> : ignore null and empty strings in the result |
| Result    | Text    | <- String containing all elements of the collection, separated by delimiter           |

## Description

The `.join()` function converts all elements of the collection to strings and concatenates them using the specified *delimiter* string as separator. The function returns the resulting string.

This function does not modify the original collection.

By default, null or empty elements of the collection are returned in the resulting string. Pass the `ck ignore null or empty` constant in the *option* parameter if you want to remove them from the resulting string.

## Example

```
var $c : Collection
var $t1;$t2 : Text
$c:=New collection(1;2;3;"Paris";Null;"";4;5)
$t1:=$c.join("|") //1|2|3|Paris|null||4|5
$t2:=$c.join("|";ck ignore null or empty) //1|2|3|Paris|4|5
```

## .last()

- History

**.last() : any**

| Parameter | Type | Description                   |
|-----------|------|-------------------------------|
| Result    | any  | <- Last element of collection |

## Description

The `.last()` function returns the last element of the collection.

This function does not modify the original collection.

The function returns `Undefined` if the collection is empty.

## Example

```
var $col; $emptyCol : Collection
var $last : Variant
$c:=New collection(10; 20; 30; "hello"; 50)
$last:=$col.last() // 50

$emptyCol:=New collection() //empty
// $last:=$emptyCol[$emptyCol.length-1] //returns an error
$last:=$emptyCol.last() // returns Undefined
```

## .lastIndexOf()

- History

**.lastIndexOf( *toSearch* : expression { ; *startFrom* : Integer } ) : Integer**

| Parameter        | Type          | Description   |
|------------------|---------------|---|
| <i>toSearch</i>  | expression -> | The element that is to be searched for within the collection                      |
| <i>startFrom</i> | Integer       | -> Index to start the search at   |
| <i>Result</i>    | Integer       | <- Index of last occurrence of <i>toSearch</i> in the collection, -1 if not found |

### Description

The `.lastIndexOf()` function searches the *toSearch* expression among collection elements and returns the index of the last occurrence, or -1 if it was not found.

This function does not modify the original collection.

In *toSearch*, pass the expression to find in the collection. You can pass:

- a scalar value (text, number, boolean, date),
- the null value,
- an object or a collection reference.

*toSearch* must match exactly the element to find (the same rules as for the equality operator are applied).

Optionally, you can pass the index of collection from which to start a reverse search in *startFrom*.

- If *startFrom*  $\geq$  the collection's length minus one (`coll.length-1`), the whole collection is searched (default).
- If *startFrom*  $< 0$ , it is recalculated as  $startFrom := startFrom + length$  (it is considered as the offset from the end of the collection). If the calculated value is negative, -1 is returned (the collection is not searched). **Note:** Even if *startFrom* is negative, the collection is still searched from right to left.
- If *startFrom* = 0, -1 is returned, which means the collection is not searched.

### Example

```
var $col : Collection
var $pos1;$pos2;$pos3;$pos4;$pos5 : Integer
$col:=Split string("a,b,c,d,e,f,g,h,i,j,e,k,e;","","") // $col.length=13
npos1:=$col.lastIndexOf("e") // returns 12
npos2:=$col.lastIndexOf("e";6) // returns 4
npos3:=$col.lastIndexOf("e";15) // returns 12
npos4:=$col.lastIndexOf("e";-2) // returns 10
npos5:=$col.lastIndexOf("x") // returns -1
```

## .length

- History

**.length : Integer**

### Description

The `.length` property returns the number of elements in the collection.

The `.length` property is initialized when the collection is created. Adding or removing elements updates the length, if necessary. This property is **read-only** (you cannot use it to set the size of the collection).

## Example

```
var $col : Collection // $col.length initialized to 0
$col:=New collection("one";"two";"three") // $col.length updated to 3
$col[4]:="five" // $col.length updated to 5
$vSize:=$col.remove(0;3).length // $vSize=2
```

## .map()

- History

`.map( formula : 4D.Function { ; ...param : any } ) : Collection`

`.map( methodName : Text { ; ...param : any } ) : Collection`

| Parameter  | Type                        | Description  |
|------------|-----------------------------|--|
| formula    | 4D.Function->Formula object |  |
| methodName | Text                        | -> Name of a method  |
| param      | any                         | -> Parameter(s) to pass to <i>formula</i> or <i>methodName</i> |
| Result     | Collection                  | <- Collection of transformed values                            |

## Description

The `.map()` function creates a new collection based upon the result of the call of the *formula* 4D function or *methodName* method on each element of the original collection. Optionally, you can pass parameters to *formula* or *methodName* using the *param* parameter(s). `.map()` always returns a collection with the same size as the original collection, except if `$1.stop` was used (see below).

This function does not modify the original collection.

You designate the callback to be executed to evaluate collection elements using either:

- *formula* (recommended syntax), a [Formula object](#) that can encapsulate any executable expressions, including functions and project methods;
- or *methodName*, the name of a project method (text).

The callback is called with the parameter(s) passed in *param* (optional). The callback can perform any operation, with or without the parameter(s) and must return new transformed value to add to the resulting collection. It receives an `Object` in first parameter (`$1`).

The callback receives the following parameters:

- in `$1.value`: element value to be evaluated
- in `$2`: param
- in `$N...`: paramN...

It can set the following parameter(s):

- (mandatory if you used a method) `$1.result` (any type): new transformed value to add to the resulting collection
- `$1.stop` (Boolean, optional): **true** to stop the method callback. The returned value is the last calculated.

## Example

```
var $c; $c2 : Collection  
$c:=New collection(1; 4; 9; 10; 20)  
$c2:=$c.map(Formula(Round(($.value/$2)*100; 2)); $c.sum())  
//$c2=[2.27,9.09,20.45,22.73,45.45]
```

## .max()

- History

**.max( { propertyPath : Text } ) : any**

| Parameter         | Type  | Description                                       |
|-------------------|---|---|
| propertyPath Text |   | -> Object property path to be used for evaluation |
| Result            | Boolean, Text, Number, Collection, Object, Date | <-Maximum value in the collection                 |

## Description

The `.max()` function returns the element with the highest value in the collection (the last element of the collection as it would be sorted in ascending order using the [.sort\(\)](#) function).

This function does not modify the original collection.

If the collection contains different types of values, the `.max()` function will return the maximum value within the last element type in the type list order (see [.sort\(\)](#) description).

If the collection contains objects, pass the *propertyPath* parameter to indicate the object property whose maximum value you want to get.

If the collection is empty, `.max()` returns *Undefined*.

## Example

```
var $col : Collection  
$col:=New collection(200;150;55)  
$col.push(New object("name";"Smith";"salary";10000))  
$col.push(New object("name";"Wesson";"salary";50000))  
$col.push(New object("name";"Alabama";"salary";10500))  
$max:=$col.max() // {name:Alabama, salary:10500}  
$maxSal:=$col.max("salary") //50000  
$maxName:=$col.max("name") //"Wesson"
```

## .min()

- History

**.min( { propertyPath : Text } ) : any**

| Parameter         | Type  | Description                                       |
|-------------------|---|---|
| propertyPath Text |   | -> Object property path to be used for evaluation |
| Result            | Boolean, Text, Number, Collection, Object, Date | <-Minimum value in the collection                 |

## Description

The `.min()` function returns the element with the smallest value in the collection (the first element of the collection as it would be sorted in ascending order using the [.sort\(\)](#) function).

This function does not modify the original collection.

If the collection contains different types of values, the `.min()` function will return the minimum value within the first element type in the type list order (see [.sort\(\)](#) description).

If the collection contains objects, pass the *propertyPath* parameter to indicate the object property whose minimum value you want to get.

If the collection is empty, `.min()` returns *Undefined*.

## Example

```
var $col : Collection
$col:=New collection(200;150;55)
$col.push(New object("name";"Smith";"salary";10000))
$col.push(New object("name";"Wesson";"salary";50000))
$col.push(New object("name";"Alabama";"salary";10500))
$min:=$col.min() //55
$minSal:=$col.min("salary") //10000
$minName:=$col.min("name") //"Alabama"
```

## .orderBy()

- History

**.orderBy( )** : Collection  
**.orderBy( pathStrings : Text )** : Collection  
**.orderBy( pathObjects : Collection )** : Collection  
**.orderBy( ascOrDesc : Integer )** : Collection

| Parameter   | Type       | Description  |
|-------------|------------|--|
| pathStrings | Text       | -> Property path(s) on which to order the collection                       |
| pathObjects | Collection | -> Collection of criteria objects  |
| ascOrDesc   | Integer    | -> <code>ck ascending</code> or <code>ck descending</code> (scalar values) |
| Result      | Collection | <- Ordered copy of the collection (shallow copy)                           |

## Description

The `.orderBy()` function returns a new collection containing all elements of the collection in the specified order.

This function returns a *shallow copy*, which means that objects or collections in both collections share the same reference. If the original collection is a shared collection, the returned collection is also a shared collection.

This function does not modify the original collection.

If you pass no parameter, the function orders scalar values in the collection in ascending order (other element types such as objects or collections are returned unordered). You can modify this automatic order by passing the `ck ascending` or `ck descending` constants in the *ascOrDesc* parameter (see below).

You can also pass a criteria parameter to define how the collection elements must be sorted. Three syntaxes are supported for this parameter:

- *pathStrings* : Text (formula). **Syntax:** `propertyPath1 {desc or asc}, propertyPath2 {desc or asc}, ...` (default order: asc). *pathStrings* contains a formula made of 1 to x property paths and (optionally) sort orders, separated by commas. The order in which the properties are passed determines the sorting priority of the collection elements. By default, properties are sorted in ascending order. You can set the sort order of a property in the criteria string, separated from the property path by a single space: pass "asc" to sort in ascending order or "desc" in descending order.
- *pathObjects* : Collection. You can add as many objects in the *pathObjects* collection as necessary. By default, properties are sorted in ascending order ("descending" is false). Each element of the collection contains an object structured in the following way:

```
{  
    "propertyPath": string,  
    "descending": boolean  
}
```

- *ascOrDesc* : Integer. You pass one of the following constants from the **Objects and collections** theme:

| Constant      | Type     | Value | Comment   |
|---------------|----------|-------|---|
| ck ascending  | Longint0 |       | Elements are ordered in ascending order (default) |
| ck descending | Longint1 |       | Elements are ordered in descending order          |

This syntax orders scalar values in the collection only (other element types such as objects or collections are returned unordered).

If the collection contains elements of different types, they are first grouped by type and sorted afterwards. Types are returned in the following order:

1. null
2. booleans
3. strings
4. numbers
5. objects
6. collections
7. dates

## Example 1

Ordering a collection of numbers in ascending and descending order:

```
var $c; $c2; $c3 : Collection  
$c:=New collection  
For ($vCounter;1;10)  
    $c.push(Random)  
End for  
$c2:=$c.orderBy(ck ascending)  
$c3:=$c.orderBy(ck descending)
```

## Example 2

Ordering a collection of objects based on a text formula with property names:

```
var $c; $c2 : Collection
$c:=New collection
For($vCounter;1;10)
    $c.push(New object("id";$vCounter;"value";Random) )
End for
$c2:=$c.orderBy("value desc")
$c2:=$c.orderBy("value desc, id")
$c2:=$c.orderBy("value desc, id asc")
```

Ordering a collection of objects with a property path:

```
var $c; $c2 : Collection
$c:=New collection
$c.push(New object("name";"Cleveland";"phones";New
object("p1";"01";"p2";"02")))
$c.push(New object("name";"Blountsville";"phones";New
object("p1";"00";"p2";"03"))

$c2:=$c.orderBy("phones.p1 asc")
```

### Example 3

Ordering a collection of objects using a collection of criteria objects:

```
var $crit; $c; $c2 : Collection
$crit:=New collection
$c:=New collection
For($vCounter;1;10)
    $c.push(New object("id";$vCounter;"value";Random) )
End for
$crit.push(New object("propertyPath";"value";"descending";True))
$crit.push(New object("propertyPath";"id";"descending";False))
$c2:=$c.orderBy($crit)
```

Ordering with a property path:

```
var $crit; $c; $c2 : Collection
$c:=New collection
$c.push(New object("name";"Cleveland";"phones";New
object("p1";"01";"p2";"02")))
$c.push(New object("name";"Blountsville";"phones";New
object("p1";"00";"p2";"03")))
$crit:=New collection(New
object("propertyPath";"phones.p2";"descending";True))
$c2:=$c.orderBy($crit)
```

## .orderByMethod()

- History

**.orderByMethod( formula : 4D.Function { ; ...extraParam : expression } ) : Collection**  
**.orderByMethod( methodName : Text { ; ...extraParam : expression } ) : Collection**

| Parameter  | Type                        | Description  |
|------------|-----------------------------|--|
| formula    | 4D.Function->Formula object |  |
| methodName | Text                        | -> Name of a method                                |
| extraParam | any                         | -> Parameter(s) to pass                            |
| Result     | Collection                  | <- Sorted copy of the collection (shallow<br>copy) |

## Description

The `.orderByMethod()` function returns a new collection containing all elements of the collection in the order defined through the *formula* 4D function or *methodName* method.

This function returns a *shallow copy*, which means that objects or collections in both collections share the same reference. If the original collection is a shared collection, the returned collection is also a shared collection.

This function does not modify the original collection.

You designate the callback to be executed to evaluate collection elements using either:

- *formula* (recommended syntax), a [Formula object](#) that can encapsulate any executable expressions, including functions and project methods;
- or *methodName*, the name of a project method (text).

In the callback, pass some code that compares two values and returns **true** if the first value is lower than the second value. You can provide *extraParam* parameters to the callback if necessary.

The callback receives the following parameters:

- `$1` (object), where:
  - `$1.value` (any type): first element value to be compared
  - `$1.value2` (any type): second element value to be compared
  - `$2...$N` (any type): extra parameters

If you used a method, it must set the following parameter:

- `$1.result` (boolean): **true** if `$1.value < $1.value2`, **false** otherwise

### Example 1

You want to sort a collection of strings in numerical order rather than alphabetical order:

```
var $c; $c2; $c3 : Collection
$c:=New collection
$c.push("33";"4";"1111";"222")
$c2:=$c.orderBy() // $c2=["1111","222","33","4"], alphabetical order
$c3:=$c.orderByMethod(Formula(Num($1.value)<Num($1.value2))) // $c3= ["4","33","222","1111"]
```

### Example 2

You want to sort a collection of strings on their length:

```
var $fruits; $c2 : Collection
$fruits:=New collection("Orange";"Apple";"Grape";"pear";"Banana";"fig";"Blackberry";
$c2:=$fruits.orderByMethod(Formula(Length(String($1.value))>Length(Str)))
// $c2=[Passion fruit, Blackberry, Orange, Banana, Apple, Grape, pear, fig]
```

### Example 3

You want to sort a collection by character code or language:

```

var $strings1; $strings2 : Collection
$strings1:=New
collection("Alpha";"Charlie";"alpha";"bravo";"Bravo";"charlie")

//using the character code:
$strings2:=$strings1.orderByMethod(Function(sortCollection);sk
character codes)
// result : ["Alpha","Bravo","Charlie","alpha","bravo","charlie"]

//using the language:
$strings2:=$strings1.orderByMethod(Function(sortCollection);sk strict)
// result : ["alpha","Alpha","bravo","Bravo","charlie","Charlie"]

```

The ***sortCollection*** method:

```

var $1 : Object
var $2: Integer // sort option

$1.result:=(Compare strings ($1.value;$1.value2;$2)<0)

```

## .pop()

- History

**.pop() :** any

| <b>Parameter Type</b> | <b>Description</b> |
|-----------------------|--------------------|
|-----------------------|--------------------|

|        |                                  |
|--------|----------------------------------|
| Result | any <-Last element of collection |
|--------|----------------------------------|

### Description

The `.pop()` function removes the last element from the collection and returns it as the function result.

This function modifies the original collection.

When applied to an empty collection, `.pop()` returns ***undefined***.

### Example

`.pop()`, used in conjunction with [`.push\(\)`](#), can be used to implement a first-in, last-out stack feature:

```

var $stack : Collection
$stack:=New collection //$/stack=[]
$stack.push(1;2) //$/stack=[1,2]
$stack.pop() //$/stack=[1] Returns 2
$stack.push(New collection(4;5)) //$/stack=[[1,[4,5]]
$stack.pop() //$/stack=[1] Returns [4,5]
$stack.pop() //$/stack=[] Returns 1

```

## .push()

- History

**.push( *element* : any { ;...*elementN* } ) :** Collection

| <b>Parameter Type</b> | <b>Description</b> |
|-----------------------|--------------------|
|-----------------------|--------------------|

|         |   |
|---------|---|
| element | Mixed ->Element(s) to add to the collection |
|---------|---|

|        |   |
|--------|---|
| Result | Collection <- Original collection containing added elements |
|--------|---|

## Description

The `.push()` function appends one or more *element(s)* to the end of the collection instance and returns the edited collection.

This function modifies the original collection.

### Example 1

```
var $col : Collection
$col:=New collection(1;2) //$/col=[1,2]
$col.push(3) //$/col=[1,2,3]
$col.push(6;New object("firstname";"John";"lastname";"Smith"))
//$/col=[1,2,3,6,{firstname:John,lastname:Smith}]
```

### Example 2

You want to sort the resulting collection:

```
var $col; $sortedCol : Collection
$col:=New collection(5;3;9) //$/col=[5,3,9]
$sortedCol:=$col.push(7;50).sort()
//$/col=[5,3,9,7,50]
//$/sortedCol=[3,5,7,9,50]
```

## .query()

- History

`.query( queryString : Text ; ...value : any ) : Collection`  
`.query( queryString : Text ; querySettings : Object ) : Collection`

| Parameter           | Type       | Description  |
|---------------------|------------|--|
| queryString         | Text       | -> Search criteria                                   |
| value               | Mixed      | -> Value(s) to compare when using placeholder(s)     |
| querySettingsObject |            | -> Query options: parameters, attributes             |
| Result              | Collection | <- Element(s) matching queryString in the collection |

## Description

The `.query()` function returns all elements of a collection of objects that match the search conditions defined by *queryString* and (optionally) *value* or *querySettings*. If the original collection is a shared collection, the returned collection is also a shared collection.

This function does not modify the original collection.

The *queryString* parameter uses the following syntax:

```
propertyPath comparator value {logicalOperator propertyPath comparator  
value}
```

For detailed information on how to build a query using *queryString*, *value* and *querySettings* parameters, please refer to the [dataClass.query\(\)](#) function description.

Formulas are not supported by the `collection.query()` function, neither in the *queryString* parameter nor as *formula* object parameter.

## Example 1

```
var $c; $c2; $c3 : Collection
$c:=New collection
$c.push(New object("name";"Cleveland";"zc";35049))
$c.push(New object("name";"Blountsville";"zc";35031))
$c.push(New object("name";"Adger";"zc";35006))
$c.push(New object("name";"Clanton";"zc";35046))
$c.push(New object("name";"Clanton";"zc";35045))
$c2:=$c.query("name = :1";"Cleveland") //$/c2=
[{name:Cleveland,zc:35049}]
$c3:=$c.query("zc > 35040") //$/c3=[{name:Cleveland,zc:35049},
{name:Clanton,zc:35046},{name:Clanton,zc:35045}]
```

## Example 2

```
var $c : Collection
$c:=New collection
$c.push(New object("name";"Smith";"dateHired";!22-05-2002!;"age";45))
$c.push(New object("name";"Wesson";"dateHired";!30-11-2017!))
$c.push(New object("name";"Winch";"dateHired";!16-05-2018!;"age";36))

$c.push(New object("name";"Sterling";"dateHired";!10-5-
1999!;"age";Null))
$c.push(New object("name";"Mark";"dateHired";!01-01-2002!))
```

This example returns persons whose name contains "in":

```
$col:=$c.query("name = :1";"@in@")
//$/col=[{name:Winch...},{name:Sterling...}]
```

This example returns persons whose name does not begin with a string from a variable (entered by the user, for example):

```
$col:=$c.query("name # :1";$aString+"@")
//if $astrig="W"
//$/col=[{name:Smith...},{name:Sterling...},{name:Mark...}]
```

This example returns persons whose age is not known (property set to null or undefined):

```
$col:=$c.query("age=null") //placeholders not allowed with "null"
//$/col=[{name:Wesson...},{name:Sterling...},{name:Mark...}]
```

This example returns persons hired more than 90 days ago:

```
$col:=$c.query("dateHired < :1";(Current date-90))
//$/col=[{name:Smith...},{name:Sterling...},{name:Mark...}] if today
is 01/10/2018
```

## Example 3

More examples of queries can be found in the [dataClass.query\(\)](#) page.

## .reduce()

- History

**.reduce( formula : 4D.Function { ; initialValue : any { ; ...param : expression } } ) : any**  
**.reduce( methodName : Text { ; initialValue : any { ; ...param : expression } } ) : any**

| Parameter  | Type  | Description  |
|------------|---|--|
| formula    | 4D.Function                                     | -> Formula object  |
| methodName | Text  | -> Name of a method  |
| initValue  | Text, Number, Object, Collection, Date, Boolean | -> Value to use as the first argument to the first call of <i>formula</i> or <i>methodName</i> |
| param      | expression                                      | -> Parameter(s) to pass  |
| Result     | Text, Number, Object, Collection, Date, Boolean | <-Result of the accumulator value  |

## Description

The `.reduce()` function applies the *formula* or *methodName* callback against an accumulator and each element in the collection (from left to right) to reduce it to a single value.

This function does not modify the original collection.

You designate the callback to be executed to evaluate collection elements using either:

- *formula* (recommended syntax), a [Formula object](#) that can encapsulate any executable expressions, including functions and project methods;
- or *methodName*, the name of a project method (text).

The callback takes each collection element and performs any desired operation to accumulate the result into `$1.accumulator`, which is returned in `$1.value`.

You can pass the value to initialize the accumulator in *initValue*. If omitted, `$1.accumulator` starts with *Undefined*.

The callback receives the following parameters:

- in `$1.value`: element value to be processed
- in `$2: param`
- in `$N...: paramN...`

The callback sets the following parameter(s):

- `$1.accumulator`: value to be modified by the function and which is initialized by *initValue*.
- `$1.stop` (boolean, optional): **true** to stop the method callback. The returned value is the last calculated.

## Example 1

```
var $c : Collection
$c:=New collection(5;3;5;1;3;4;4;6;2;2)
$r:=$c.reduce(Formula($1.accumulator*=$1.value); 1) //returns 86400
```

## Example 2

This example allows reducing several collection elements to a single one:

```
var $c;$r : Collection
$c:=New collection
$c.push(New collection(0;1))
$c.push(New collection(2;3))
$c.push(New collection(4;5))
$c.push(New collection(6;7))
$r:=$c.reduce(Formula(Flatten)) //r=[0,1,2,3,4,5,6,7]
```

With the following ***Flatten*** method:

```
If ($1.accumulator=NULL)
    $1.accumulator:=New collection
End if
$1.accumulator.combine($1.value)
```

## .reduceRight()

- History

**.reduceRight( *formula* : 4D.Function { ; *initValue* : any { ; ...*param* : expression } } )**  
: any  
**.reduceRight( *methodName* : Text { ; *initValue* : any { ; ...*param* : expression } } )** : any

| Parameter  | Type  | Description  |
|------------|---|--|
| formula    | 4D.Function                                     | -> Formula object  |
| methodName | Text  | -> Name of a method  |
| initValue  | Text, Number, Object, Collection, Date, Boolean | -> Value to use as the first argument to the first call of <i>formula</i> or <i>methodName</i> |
| param      | expression                                      | -> Parameter(s) to pass  |
| Result     | Text, Number, Object, Collection, Date, Boolean | <- Result of the accumulator value   |

### Description

The `.reduceRight()` function applies the *formula* or *methodName* callback against an accumulator and each element in the collection (from right to left) to reduce it to a single value.

This function does not modify the original collection.

You designate the callback to be executed to evaluate collection elements using either:

- *formula* (recommended syntax), a [Formula object](#) that can encapsulate any executable expressions, including functions and project methods;
- or *methodName*, the name of a project method (text).

The callback takes each collection element and performs any desired operation to accumulate the result into `$1.accumulator`, which is returned in `$1.value`.

You can pass the value to initialize the accumulator in *initValue*. If omitted, `$1.accumulator` starts with *Undefined*.

The callback receives the following parameters:

- in `$1.value`: element value to be processed
- in `$2`: *param*
- in `$N...`: *paramN...*

The callback sets the following parameter(s):

- `$1.accumulator`: value to be modified by the function and which is initialized by *initValue*.
- `$1.stop` (boolean, optional): **true** to stop the method callback. The returned value is the last calculated.

### Example 1

```

var $c : Collection
$c:=New collection(5;3;5;1;3;4;4;6;2;2)
$r:=$c.reduceRight(Formula($1.accumulator*=$1.value); 1) //returns
86400

```

## Example 2

This example allows reducing several collection elements to a single one:

```

var $c;$r : Collection
$c:=New collection
$c.push(New collection(0;1))
$c.push(New collection(2;3))
$c.push(New collection(4;5))
$c.push(New collection(6;7))
$r:=$c.reduceRight(Formula(Flatten)) //r=[6,7,4,5,2,3,0,1]

```

With the following ***Flatten*** method:

```

//Flatten project method
If($1.accumulator=NULL)
    $1.accumulator:=New collection
End if
$1.accumulator.combine($1.value)

```

## .remove()

- History

**.remove( *index* : Integer { ; *howMany* : Integer } ) : Collection**

| Parameter      | Type       | Description  |
|----------------|------------|--|
| <i>index</i>   | Integer    | -> Element at which to start removal                     |
| <i>howMany</i> | Integer    | -> Number of elements to remove, or 1 element if omitted |
| Result         | Collection | <-Original collection without removed element(s)         |

### Description

The `.remove()` function removes one or more element(s) from the specified *index* position in the collection and returns the edited collection.

This function modifies the original collection.

In *index*, pass the position where you want the element to be removed from the collection.

**Warning:** Keep in mind that collection elements are numbered from 0. If *index* is greater than the length of the collection, actual starting index will be set to the length of the collection.

- If *index* < 0, it is recalculated as *index*:=*index*+*length* (it is considered as the offset from the end of the collection).
- If the calculated value < 0, *index* is set to 0.
- If the calculated value > the length of the collection, *index* is set to the length.

In *howMany*, pass the number of elements to remove from *index*. If *howMany* is not specified, then one element is removed.

If you try to remove an element from an empty collection, the method does nothing (no error is generated).

## Example

```
var $col : Collection
$col:=New collection("a";"b";"c";"d";"e";"f";"g";"h")
$col.remove(3) // $col=["a","b","c","e","f","g","h"]
$col.remove(3;2) // $col=["a","b","c","g","h"]
$col.remove(-8;1) // $col=["b","c","g","h"]
$col.remove(-3;1) // $col=["b","g","h"]
```

## .resize()

- History

**.resize( size : Integer { ; defaultValue : any } ) : Collection**

| Parameter    | Type  | Description                           |
|--------------|---|---------------------------------------|
| size         | Integer   | -> New size of the collection         |
| defaultValue | Number, Text, Object, Collection, Date, Boolean | -> Default value to fill new elements |
| Result       | Collection                                      | <- Resized original collection        |

### Description

The `.resize()` function sets the collection length to the specified new size and returns the resized collection.

This function modifies the original collection.

- If `size < collection length`, exceeding elements are removed from the collection.
- If `size > collection length`, the collection length is increased to `size`.

By default, new elements are filled will **null** values. You can specify the value to fill in added elements using the `defaultValue` parameter.

## Example

```
var $c : Collection
$c:=New collection
$c.resize(10) // $c=
[null,null,null,null,null,null,null,null,null,null]

$c:=New collection
$c.resize(10;0) // $c=[0,0,0,0,0,0,0,0,0,0]

$c:=New collection(1;2;3;4;5)
$c.resize(10;New object("name";"X")) // $c=[1,2,3,4,5,{name:X},
{name:X},{name:X},{name:X},{name:X}]

$c:=New collection(1;2;3;4;5)
$c.resize(2) // $c=[1,2]
```

## .reverse()

- History

**.reverse( ) : Collection**

| <b>Parameter</b> | <b>Type</b> | <b>Description</b>                 |
|------------------|-------------|------------------------------------|
| Result           | Collection  | <- Inverted copy of the collection |

## Description

The `.reverse()` function returns a deep copy of the collection with all its elements in reverse order. If the original collection is a shared collection, the returned collection is also a shared collection.

This function does not modify the original collection.

## Example

```
var $c; $c2 : Collection
$c:=New collection(1;3;5;2;4;6)
$c2:=$c.reverse() // $c2=[6,4,2,5,3,1]
```

## .shift()

- History

**.shift()** : any

### Parameter Type Description

|        |     |                                |
|--------|-----|--------------------------------|
| Result | any | <- First element of collection |
|--------|-----|--------------------------------|

## Description

The `.shift()` function removes the first element of the collection and returns it as the function result.

This function modifies the original collection.

If the collection is empty, this method does nothing.

## Example

```
var $c : Collection
var $val : Variant
$c:=New collection(1;2;4;5;6;7;8)
$val:=$c.shift()
// $val=1
// $c=[2,4,5,6,7,8]
```

## .slice()

- History

**.slice( startFrom : Integer { ; end : Integer } )** : Collection

### Parameter Type Description

|           |            |   |
|-----------|------------|---|
| startFrom | Integer    | -> Start index (included)                                   |
| end       | Integer    | -> End index (not included)                                 |
| Result    | Collection | <- New collection containing sliced elements (shallow copy) |

## Description

The `.slice()` function returns a portion of a collection into a new collection, selected from *startFrom* index to *end* index (*end* not included). This function returns a *shallow copy* of the collection. If the original collection is a shared collection, the returned collection is also a shared collection.

This function does not modify the original collection.

The returned collection contains the element specified by *startFrom* and all subsequent elements up to, but not including, the element specified by *end*. If only the *startFrom* parameter is specified, the returned collection contains all elements from *startFrom* to the last element of the original collection.

- If *startFrom* < 0, it is recalculated as *startFrom*:=*startFrom*+*length* (it is considered as the offset from the end of the collection).
- If the calculated value < 0, *startFrom* is set to 0.
- If *end* < 0 , it is recalculated as *end*:=*end*+*length*.
- If *end* < *startFrom* (passed or calculated values), the method does nothing.

## Example

```
var $c; $nc : Collection  
$c:=New collection(1;2;3;4;5)  
$nc:=$c.slice(0;3) // $nc=[1,2,3]  
$nc:=$c.slice(3) // $nc=[4,5]  
$nc:=$c.slice(1;-1) // $nc=[2,3,4]  
$nc:=$c.slice(-3;-2) // $nc=[3]
```

## .some()

### ▪ History

`.some( { startFrom : Integer ; } formula : 4D.Function { ; ...param : any } ) :`  
`Boolean`  
`.some( { startFrom : Integer ; } methodName : Text { ; ...param : any } ) : Boolean`

| Parameter  | Type        | Description  |
|------------|-------------|--|
| startFrom  | Integer     | -> Index to start the test at                                |
| formula    | 4D.Function | -> Formula object  |
| methodName | Text        | -> Name of a method  |
| param      | Mixed       | -> Parameter(s) to pass                                      |
| Result     | Boolean     | <- True if at least one element successfully passed the test |

## Description

The `.some()` function returns true if at least one element in the collection successfully passed a test implemented in the provided *formula* or *methodName* code.

You designate the 4D code (callback) to be executed to evaluate collection elements using either:

- *formula* (recommended syntax), a [Formula object](#) that can encapsulate any executable expressions, including functions and project methods;
- or *methodName*, the name of a project method (text).

The callback is called with the parameter(s) passed in *param* (optional). The callback can perform any test, with or without the parameter(s) and must return **true** for every element fulfilling the test. It receives an `Object` in first parameter (\$1).

The callback receives the following parameters:

- in `$1.value`: element value to be processed
- in `$2: param`
- in `$N...: paramN...`

It can set the following parameter(s):

- (mandatory if you used a method) `$1.result` (boolean): **true** if the element value evaluation is successful, **false** otherwise.
- `$1.stop` (boolean, optional): **true** to stop the method callback. The returned value is the last calculated.

In any case, at the point where `.some()` function encounters the first collection element returning true, it stops calling the callback and returns **true**.

By default, `.some()` tests the whole collection. Optionally, you can pass the index of an element from which to start the test in `startFrom`.

- If `startFrom >=` the collection's length, **False** is returned, which means the collection is not tested.
- If `startFrom < 0`, it is considered as the offset from the end of the collection.
- If `startFrom = 0`, the whole collection is searched (default).

## Example

You want to know if at least one collection value is `>0`.

```
var $c : Collection
var $b : Boolean
$c:=New collection
$c.push(-5;-3;-1;-4;-6;-2)
$b:=$c.some(Formula($1.value>0)) // $b=false
$c.push(1)
$b:=$c.some(Formula($1.value>0)) // $b=true

$c:=New collection
$c.push(1;-5;-3;-1;-4;-6;-2)
$b:=$c.some(Formula($1.value>0)) // $b=true
$b:=$c.some(1;Formula($1.value>0)) // $b=false
```

## .sort()

### ■ History

`.sort( formula : 4D.Function { ; ...extraParam : any } ) : Collection`  
`.sort( methodName : Text { ; ...extraParam : any } ) : Collection`

| Parameter  | Type                        | Description                    |
|------------|-----------------------------|--------------------------------|
| formula    | 4D.Function->Formula object |                                |
| methodName | Text                        | -> Name of a method            |
| extraParam | any                         | -> Parameter(s) for the method |
| Result     | Collection                  | <-Original collection sorted   |

## Description

The `.sort()` function sorts the elements of the original collection and also returns the sorted collection .

This function modifies the original collection.

If `.sort()` is called with no parameters, only scalar values (number, text, date, booleans) are sorted. Elements are sorted by default in ascending order, according to their type.

If you want to sort the collection elements in some other order or sort any type of element, you must supply in *formula* ([Formula object](#)) or *methodName* (Text) a comparison callback that compares two values and returns **true** if the first value is lower than the second value. You can provide additional parameters to the callback if necessary.

The callback receives the following parameters:

- `$1` (object), where:
  - `$1.value` (any type): first element value to be compared
  - `$1.value2` (any type): second element value to be compared
- `$2...$N` (any type): extra parameters

If you used a method, you must set the following parameter:

- `$1.result` (boolean): **true** if `$1.value < $1.value2`, **false** otherwise.

If the collection contains elements of different types, they are first grouped by type and sorted afterwards. Types are returned in the following order:

1. null
2. booleans
3. strings
4. numbers
5. objects
6. collections
7. dates

### Example 1

```
var $col; $col2 : Collection
$col:=New
collection("Tom";5;"Mary";3;"Henry";1;"Jane";4;"Artie";6;"Chip";2)
$col2:=$col.sort() // $col2=
["Artie","Chip","Henry","Jane","Mary","Tom",1,2,3,4,5,6]
// $col=["Artie","Chip","Henry","Jane","Mary","Tom",1,2,3,4,5,6]
```

### Example 2

```
var $col; $col2 : Collection
$col:=New collection(10;20)
$col2:=$col.push(5;3;1;4;6;2).sort() // $col2=[1,2,3,4,5,6,10,20]
```

### Example 3

```
var $col; $col2; $col3 : Collection
$col:=New collection(33;4;66;1111;222)
$col2:=$col.sort() // numerical sort: [4,33,66,222,1111]
$col3:=$col.sort(Formula(String($1.value)<String($1.value2)))
// alphabetical sort: [1111,222,33,4,66]
```

## .sum()

- History

**.sum( { propertyPath : Text } ) : Real**

| Parameter    | Type | Description  |
|--------------|------|--|
| propertyPath | Text | -> Object property path to be used for calculation |
| Result       | Real | <-Sum of collection values                         |

### Description

The `.sum()` function returns the sum for all values in the collection instance.

Only numerical elements are taken into account for the calculation (other element types are ignored).

If the collection contains objects, pass the *propertyPath* parameter to indicate the object property to take into account.

`.sum()` returns 0 if:

- the collection is empty,
- the collection does not contain numerical elements,
- *propertyPath* is not found in the collection.

### Example 1

```
var $col : Collection
var $vSum : Real
$col:=New collection(10;20;"Monday";True;2)
$vSum:=$col.sum() //32
```

### Example 2

```
var $col : Collection
var $vSum : Real
$col:=New collection
$col.push(New object("name";"Smith";"salary";10000))
$col.push(New object("name";"Wesson";"salary";50000))
$col.push(New object("name";"Gross";"salary";10500,5))
$vSum:=$col.sum("salary") //vSum=70500,5
```

## .unshift()

- History

**.unshift( value : any { ;...valueN : any } ) : Collection**

| Parameter | Type                                   | Description  |
|-----------|--|--|
| value     | Text, Number, Object, Collection, Date | -> Value(s) to insert at the beginning of the collection |
| Result    | Real                                   | <-Collection containing added element(s)                 |

|

### Description

The `.unshift()` function inserts the given *value(s)* at the beginning of the collection and returns the modified collection.

This function modifies the original collection.

If several values are passed, they are inserted all at once, which means that they appear in the resulting collection in the same order as in the argument list.

### **Example**

```
var $c : Collection
$c:=New collection(1;2)
$c.unshift(4) // $c=[4,1,2]
$c.unshift(5) // $c=[5,4,1,2]
$c.unshift(6;7) // $c=[6,7,5,4,1,2]
```

# CryptoKey

The `CryptoKey` class in the 4D language encapsulates an asymmetric encryption key pair.

This class is available from the `4D` class store.

## Example

The following sample code signs and verifies a message using a new ECDSA key pair, for example in order to make a ES256 JSON Web token.

```
// Generate a new ECDSA key pair
$key:=4D.CryptoKey.new(New
object("type";"ECDSA";"curve";"prime256v1"))

// Get signature as base64
$message:="hello world"
$signature:=$key.sign($message;New object("hash";"SHA256"))

// Verify signature
$status:=$key.verify($message;$signature;New object("hash";"SHA256"))
ASSERT($status.success)
```

## Summary

**4D.CryptoKey.new( settings : Object ) : 4D.CryptoKey** creates a new `4D.CryptoKey` object encapsulating an encryption key pair  
**.curve : Text** normalised curve name of the key  
**.decrypt( message : Text ; options : Object ) : Object** decrypts the *message* parameter using the **private** key  
**.encrypt( message : Text ; options : Object ) : Text** encrypts the *message* parameter using the **public** key  
**.getPrivateKey() : Text** returns the private key of the `CryptoKey` object  
**.getPublicKey( ) : Text** returns the public key of the `CryptoKey` object  
**.sign (message : Text ; options : Text) : Text** signs the utf8 representation of a message string  
**.size : Integer** the size of the key in bits  
**.type : Text** name of the key type - "RSA", "ECDSA", "PEM"  
**.verify( message : Text ; signature : Text ; options : Object ) : object** verifies the base64 signature against the utf8 representation of *message*

## 4D.CryptoKey.new()

- History

**4D.CryptoKey.new( settings : Object ) : 4D.CryptoKey**

| Parameter | Type           | Description                                 |
|-----------|----------------|---|
| settings  | Object         | -> Settings to generate or load a key pair  |
| result    | 4D.CryptoKey<- | Object encapsulating an encryption key pair |

The `4D.CryptoKey.new()` function creates a new `4D.CryptoKey` object encapsulating an encryption key pair, based upon the *settings* object parameter. It allows to generate a new RSA or ECDSA key, or to load an existing key pair from a

PEM definition.

### ***settings***

| <b>Property</b> | <b>Type</b> | <b>Description</b>  |
|-----------------|-------------|---|
|                 |             | Defines the type of the key to create:  |
| <u>type</u>     | text        | <ul style="list-style-type: none"><li>• "RSA": generates a RSA key pair, using <u>.size</u> as size.</li><li>• "ECDSA": generates an Elliptic Curve Digital Signature Algorithm key pair, using <u>.curve</u> as curve. Note that ECDSA keys cannot be used for encryption but only for signature.</li><li>• "PEM": loads a key pair definition in PEM format, using <u>.pem</u>.</li></ul> |
| <u>curve</u>    | text        | Name of ECDSA curve   |
| <u>pem</u>      | text        | PEM definition of an encryption key to load   |
| <u>size</u>     | integer     | Size of RSA key in bits   |

### ***CryptoKey***

The returned `CryptoKey` object encapsulates an encryption key pair. It is a shared object and can therefore be used by multiple 4D processes simultaneously.

### **.curve**

- History

#### **.curve : Text**

Defined only for ECDSA keys: the normalised curve name of the key. Usually "prime256v1" for ES256 (default), "secp384r1" for ES384, "secp521r1" for ES512.

### **.decrypt()**

- History

#### **.decrypt( message : Text ; options : Object ) : Object**

| <b>Parameter</b> | <b>Type</b> | <b>Description</b>  |
|------------------|-------------|---|
| message          | Text        | -> Message string to be decoded using <u>options.encodingEncrypted</u> and decrypted. |
| options          | Object      | -> Decoding options   |
| Result           | Object      | <- Status   |

The `.decrypt()` function decrypts the *message* parameter using the **private** key. The algorithm used depends on the type of the key.

The key must be a RSA key, the algorithm is RSA-OAEP (see [RFC 3447](#)).

### ***options***

| Property               | Type | Description   |
|------------------------|------|---|
| hash                   | text | Digest algorithm to use. For example: "SHA256", "SHA384", or "SHA512".  |
| encodingEncrypted text |      | Encoding used to convert the <code>message</code> parameter into the binary representation to decrypt. Can be "Base64" or "Base64URL". Default is "Base64". |
| encodingDecrypted text |      | Encoding used to convert the binary decrypted message into the result string. Can be "UTF-8", "Base64", or "Base64URL". Default is "UTF-8".                 |

## Result

The function returns a status object with `success` property set to `true` if the `message` could be successfully decrypted.

| Property | Type       | Description  |
|----------|------------|--|
| success  | boolean    | True if the message has been successfully decrypted                                |
| result   | text       | Message decrypted and decoded using the <code>options.encodingDecrypted</code>     |
| errors   | collection | If <code>success</code> is <code>false</code> , may contain a collection of errors |

In case the `message` couldn't be decrypted because it was not encrypted with the same key or algorithm, the `status` object being returned contains an error collection in `status.errors`.

## .encrypt()

- History

`.encrypt( message : Text ; options : Object ) : Text`

| Parameter | Type   | Description   |
|-----------|--------|---|
| message   | Text   | -> Message string to be encoded using <code>options.encodingDecrypted</code> and encrypted. |
| options   | Object | -> Encoding options   |
| Result    | Text   | <- Message encrypted and encoded using the <code>options.encodingEncrypted</code>           |

The `.encrypt()` function encrypts the `message` parameter using the **public** key. The algorithm used depends on the type of the key.

The key must be a RSA key, the algorithm is RSA-OAEP (see [RFC 3447](#)).

## *options*

| Property               | Type | Description  |
|------------------------|------|--|
| hash                   | text | Digest algorithm to use. For example: "SHA256", "SHA384", or "SHA512".   |
| encodingEncrypted text |      | Encoding used to convert the binary encrypted message into the result string. Can be "Base64", or "Base64URL". Default is "Base64".                                  |
| encodingDecrypted text |      | Encoding used to convert the <code>message</code> parameter into the binary representation to encrypt. Can be "UTF-8", "Base64", or "Base64URL". Default is "UTF-8". |

## **Result**

The returned value is an encrypted message.

### **.getPrivateKey()**

- History

**.getPrivateKey()** : Text

| <b>Parameter Type</b> | <b>Description</b> |
|-----------------------|--------------------|
|-----------------------|--------------------|

|        |                                      |
|--------|--------------------------------------|
| Result | Text <- Private key in PEM<br>format |
|--------|--------------------------------------|

The `.getPrivateKey()` function returns the private key of the `CryptoKey` object in PEM format, or an empty string if none is available.

## **Result**

The returned value is the private key.

### **.getPublicKey()**

- History

**.getPublicKey()** : Text

| <b>Parameter Type</b> | <b>Description</b> |
|-----------------------|--------------------|
|-----------------------|--------------------|

|        |                                     |
|--------|-------------------------------------|
| Result | Text <- Public key in PEM<br>format |
|--------|-------------------------------------|

The `.getPublicKey()` function returns the public key of the `CryptoKey` object in PEM format, or an empty string if none is available.

## **Result**

The returned value is the public key.

### **.pem**

- History

**.pem** : Text

PEM definition of an encryption key to load. If the key is a private key, the RSA or ECDSA public key will be deduced from it.

### **.sign()**

- History

**.sign** (*message* : Text ; *options* : Text) : Text

| <b>Parameter Type</b> | <b>Description</b>  |
|-----------------------|---|
| message               | Text -> Message string to sign  |
| options               | Object -> Signing options   |
| Result                | Text <- Signature in Base64 or Base64URL representation, depending on "encoding" option |

The `.sign()` function signs the utf8 representation of a *message* string using the `CryptoKey` object keys and provided *options*. It returns its signature in base64 or base64URL format, depending on the value of the `options.encoding` attribute you passed.

The `CryptoKey` must contain a valid **private** key.

### ***options***

| Property | Type    | Description  |
|----------|---------|--|
| hash     | text    | Digest algorithm to use. For example: "SHA256", "SHA384", or "SHA512". When used to produce a JWT, the hash size must match the PS@, ES@, RS@, or PS@ algorithm size |
| encoding | text    | Encoding used to convert the binary encrypted message into the result string. Can be "Base64", or "Base64URL". Default is "Base64".                                  |
| pss      | boolean | Use Probabilistic Signature Scheme (PSS). Ignored if the boolean key is not an RSA key. Pass <code>true</code> when producing a JWT for PS@ algorithm                |
| encoding | text    | Representation to be used for result signature. Possible values: "Base64" or "Base64URL". Default is "Base64".   |

### ***Result***

The utf8 representation of the *message* string.

## **.size**

- History

### **.size : Integer**

Defined only for RSA keys: the size of the key in bits. Typically 2048 (default).

## **.type**

- History

### **.type : Text**

Contains the name of the key type - "RSA", "ECDSA", "PEM" .

- "RSA": an RSA key pair, using `settings.size` as [.size](#).
- "ECDSA": an Elliptic Curve Digital Signature Algorithm key pair, using `settings.curve` as [.curve](#). Note that ECDSA keys cannot be used for encryption but only for signature.
- "PEM": a key pair definition in PEM format, using `settings.pem` as [.pem](#).

## **.verify()**

- History

### **.verify( *message* : Text ; *signature* : Text ; *options* : Object ) : object**

| <b>Parameter</b> | <b>Type</b> | <b>Description</b>  |
|------------------|-------------|---|
| message          | Text        | -> Message string that was used to produce the signature  |
| signature        | Text        | -> Signature to verify, in Base64 or Base64URL representation, depending on <code>options.encoding</code> value |
| options          | Object      | -> Signing options  |
| Result           | Object      | <- Status of the verification   |

The `.verify()` function verifies the base64 signature against the utf8 representation of `message` using the `CryptoKey` object keys and provided `options`.

The `CryptoKey` must contain a valid **public** key.

### ***options***

| <b>Property</b> | <b>Type</b> | <b>Description</b>   |
|-----------------|-------------|--|
| hash            | text        | Digest algorithm to use. For example: "SHA256", "SHA384", or "SHA512". When used to produce a JWT, the hash size must match the PS@, ES@, RS@, or PS@ algorithm size |
| pss             | boolean     | Use Probabilistic Signature Scheme (PSS). Ignored if the key is not an RSA key. Pass <code>true</code> when verifying a JWT for PS@ algorithm                        |
| encoding        | text        | Representation of provided signature. Possible values are "Base64" or "Base64URL". Default is "Base64".  |

### ***Result***

The function returns a status object with `success` property set to `true` if `message` could be successfully verified (i.e. the signature matches).

In case the signature couldn't be verified because it was not signed with the same `message`, key or algorithm, the `status` object being returned contains an error collection in `status.errors`.

| <b>Property</b> | <b>Type</b> | <b>Description</b>   |
|-----------------|-------------|--|
| success         | boolean     | True if the signature matches the message  |
| errors          | collection  | If <code>success</code> is <code>false</code> , may contain a collection of errors |

# DataClass

A [DataClass](#) provides an object interface to a database table. All dataclasses in a 4D application are available as a property of the `ds datastore`.

## Summary

[\*\*.attributeName\*\*](#) : [DataClassAttribute](#) objects that are available directly as properties  
[\*\*.all\*\*](#) ( { *settings* : Object } ) : [4D.EntitySelection](#) queries the datastore to find all the entities related to the dataclass and returns them as an entity selection  
[\*\*.clearRemoteCache\(\)\*\*](#) empties the ORDA cache of a dataclass  
[\*\*.fromCollection\*\*](#)( *objectCol* : Collection { ; *settings* : Object } ) : [4D.EntitySelection](#) updates or creates entities in the dataclass according to the *objectCol* collection of objects, and returns the corresponding entity selection  
[\*\*.get\*\*](#)( *primaryKey* : Integer { ; *settings* : Object } ) : [4D.Entity](#)  
[\*\*.get\*\*](#)( *primaryKey* : Text { ; *settings* : Object } ) : [4D.Entity](#) queries the dataclass to retrieve the entity matching the *primaryKey* parameter  
[\*\*.getCount\(\)\*\*](#) : Integer returns the number of entities in a dataclass  
[\*\*.getDataStore\(\)\*\*](#) : [cs.DataStore](#) returns the datastore for the specified dataclass  
[\*\*.getInfo\(\)\*\*](#) : Object returns an object providing information about the dataclass  
[\*\*.getRemoteCache\(\)\*\*](#) : Object returns an object that holds the contents of the ORDA cache for a dataclass.  
[\*\*.new\(\)\*\*](#) : [4D.Entity](#) creates in memory and returns a new blank entity related to the Dataclass  
[\*\*.newSelection\*\*](#)( { *keepOrder* : Integer } ) : [4D.EntitySelection](#) creates a new, blank, non-shareable entity selection, related to the dataclass, in memory  
[\*\*.query\*\*](#)( *queryString* : Text { ; ...*value* : any } { ; *querySettings* : Object } ) : [4D.EntitySelection](#)  
[\*\*.query\*\*](#)( *formula* : Object { ; *querySettings* : Object } ) : [4D.EntitySelection](#) searches for entities that meet the search criteria specified in *queryString* or *formula* and (optionally) *value*(s)  
[\*\*.setRemoteCacheSettings\*\*](#)(*settings* : Object) sets the timeout and maximum size of the ORDA cache for a dataclass.

## **.attributeName**

- History

**.attributeName** : [DataClassAttribute](#)

### Description

The attributes of dataclasses are objects that are available directly as properties of these classes.

The returned objects are of the [DataClassAttribute](#) class. These objects have properties that you can read to get information about your dataclass attributes.

Dataclass attribute objects can be modified, but the underlying database structure will not be altered.

### Example 1

```

$salary:=ds.Employee.salary //returns the salary attribute in the
Employee dataclass
$compCity:=ds.Company["city"] //returns the city attribute in the
Company dataclass

```

## Example 2

Considering the following database structure:

```

var $firstnameAtt;$employerAtt;$employeesAtt : Object

$firstnameAtt:=ds.Employee.firstname
//{name:firstname,kind:storage,fieldType:0,type:string,fieldNumber:2,in
//keyWordIndexed:false,autoFilled:false,mandatory:false,unique:false}

$employerAtt:=ds.Employee.employer
//{name:employer,kind:relatedEntity,relatedDataClass:Company,
//fieldType:38,type:Company,inverseName:employees}
//38=Is object

$employeesAtt:=ds.Company.employees
//{name:employees,kind:relatedEntities,relatedDataClass:Employee,
//fieldType:42,type:EmployeeSelection,inverseName:employer}
//42=Is collection

```

## Example 3

Considering the following table properties:

```

var $sequenceNumberAtt : Object
$sequenceNumberAtt=ds.Employee.sequenceNumber
//{name:sequenceNumber,kind:storage,fieldType:0,type:string,fieldNumb
//indexed:true,keyWordIndexed:false,autoFilled:true,mandatory:false,uni

```

## .all()

- History

**.all ( { settings : Object } ) : 4D.EntitySelection**

| Parameter | Type               | Description  |
|-----------|--------------------|--|
| settings  | Object             | -> Build option: context                               |
| Result    | 4D.EntitySelection | <- References on all entities related to the Dataclass |

### Description

The `.all()` function queries the datastore to find all the entities related to the dataclass and returns them as an entity selection.

The entities are returned in the default order, which is initially the order in which they were created. Note however that, if entities have been deleted and new ones added, the default order does not reflect the creation order anymore.

If no corresponding entity is found, an empty entity selection is returned.

Lazy loading is applied.

## settings

In the optional *settings* parameter, you can pass an object containing additional options. The following property is supported:

| Property Type | Description  |
|---------------|--|
| context Text  | Label for the optimization context applied to the entity selection. This context will be used by the code that handles the entity selection so that it can benefit from the optimization. This feature is <a href="#">designed for ORDA client/server processing</a> . |

To know the total number of entities in a dataclass, it is recommended to use the `getCount()` function which is more optimized than the `ds.myClass.all().length` expression.

## Example

```
var $allEmp : cs.EmployeeSelection  
$allEmp:=ds.Employee.all()
```

## .clearRemoteCache()

- History

### .clearRemoteCache()

| Parameter Type | Description                     |
|----------------|---------------------------------|
|                | Does not require any parameters |

## Description

The `.clearRemoteCache()` function empties the ORDA cache of a dataclass.

This function does not reset the `timeout` and `maxEntries` values.

## Example

```

var $ds : 4D.DataStoreImplementation
var $persons : cs.PersonsSelection
var $p : cs.PersonsEntity
var $cache : Object
var $info : Collection
var $text : Text

$ds:=Open datastore(New object ("hostname"; "www.myserver.com"); "myDS")

$persons:=$ds.Persons.all()
$text:=""
For each ($p; $persons)
    $text:=$p.firstname+" lives in "+$p.address.city+" / "
End for each

$cache:=$ds.Persons.getRemoteCache()

$ds.Persons.clearRemoteCache()
// Cache of the Persons dataclass =
{timeout:30;maxEntries:30000;stamp:255;entries:[]}

```

## .fromCollection()

- History

**.fromCollection( *objectCol* : Collection { ; *settings* : Object } )** : 4D.EntitySelection

| Parameter        | Type               | Description   |
|------------------|--------------------|---|
| <i>objectCol</i> | Collection         | -> Collection of objects to be mapped with entities |
| <i>settings</i>  | Object             | -> Build option: context                            |
| Result           | 4D.EntitySelection | <- Entity selection filled from the collection      |

### Description

The `.fromCollection()` function updates or creates entities in the dataclass according to the *objectCol* collection of objects, and returns the corresponding entity selection.

In the *objectCol* parameter, pass a collection of objects to create new or update existing entities of the dataclass. The property names must be the same as attribute names in the dataclass. If a property name does not exist in the dataclass, it is ignored. If an attribute value is not defined in the collection, its value is null.

The mapping between the objects of the collection and the entities is done on the **attribute names** and **matching types**. If an object's property has the same name as an entity's attribute but their types do not match, the entity's attribute is not filled.

### Create or update mode

For each object of *objectCol*:

- If the object contains a boolean property "`__NEW`" set to false (or does not contain a boolean "`__NEW`" property), the entity is updated or created with the corresponding values of the properties from the object. No check is performed in regards to the primary key:
  - If the primary key is given and exists, the entity is updated. In this case, the primary key can be given as is or with a "`__KEY`" property (filled with the primary key value).

- If the primary key is given (as is) and does not exist, the entity is created
- If the primary key is not given, the entity is created and the primary key value is assigned with respect to standard database rules.
- If the object contains a boolean property "\_\_NEW" set to **true**, the entity is created with the corresponding values of the attributes from the object. A check is performed in regards to the primary key:
  - If the primary key is given (as is) and exists, an error is sent
  - If the primary key is given (as is) and does not exist, the entity is created
  - If the primary is not given, the entity is created and the primary key value is assigned with respect to standard database rules.

The "\_\_KEY" property containing a value is taken into account only when the "\_\_NEW" property is set to **false** (or is omitted) and a corresponding entity exists. In all other cases, the "\_\_KEY" property value is ignored, primary key value must be passed "as is".

## Related entities

The objects of *objectCol* may contain one or more nested object(s) featuring one or more related entities, which can be useful to create or update links between entities.

The nested objects featuring related entities must contain a "\_\_KEY" property (filled with the primary key value of the related entity) or the primary key attribute of the related entity itself. The use of a \_\_KEY property allows independence from the primary key attribute name.

The content of the related entities cannot be created / updated through this mechanism.

## Stamp

If a \_\_STAMP attribute is given, a check is performed with the stamp in the datastore and an error can be returned ("Given stamp does not match current one for record# XX of table XXXX"). For more information, see [Entity locking](#).

## settings

In the optional *settings* parameter, you can pass an object containing additional options. The following property is supported:

| Property Type | Description  |
|---------------|--|
| context Text  | Label for the optimization context applied to the entity selection. This context will be used by the code that handles the entity selection so that it can benefit from the optimization. This feature is <a href="#">designed for ORDA client/server processing</a> . |

## Example 1

We want to update an existing entity. The \_\_NEW property is not given, the employee primary key is given and exists:

```

var $empsCollection : Collection
var $emp : Object
var $employees : cs.EmployeeSelection

$empsCollection:=New collection
$emp:=New object
$emp.ID:=668 //Existing PK in Employee table
$emp.firstName:="Arthur"
$emp.lastName:="Martin"
$emp.employer:=New object("ID";121) //Existing PK in the related
dataClass Company
// For this employee, we can change the Company by using another
existing PK in the related dataClass Company
$empsCollection.push($emp)
$employees:=ds.Employee.fromCollection($empsCollection)

```

## **Example 2**

We want to update an existing entity. The `__NEW` property is not given, the employee primary key is with the `__KEY` attribute and exists:

```

var $empsCollection : Collection
var $emp : Object
var $employees : cs.EmployeeSelection

$empsCollection:=New collection
$emp:=New object
$emp.__KEY:=1720 //Existing PK in Employee table
$emp.firstName:="John"
$emp.lastName:="Boorman"
$emp.employer:=New object("ID";121) //Existing PK in the related
dataClass Company
// For this employee, we can change the Company by using another
existing PK in the related dataClass Company
$empsCollection.push($emp)
$employees:=ds.Employee.fromCollection($empsCollection)

```

## **Example 3**

We want to simply create a new entity from a collection:

```

var $empsCollection : Collection
var $emp : Object
var $employees : cs.EmployeeSelection

$empsCollection:=New collection
$emp:=New object
$emp.firstName:="Victor"
$emp.lastName:="Hugo"
$empsCollection.push($emp)
$employees:=ds.Employee.fromCollection($empsCollection)

```

## **Example 4**

We want to create an entity. The `__NEW` property is True, the employee primary key is not given:

```

var $empsCollection : Collection
var $emp : Object
var $employees : cs.EmployeeSelection

$empsCollection:=New collection
$emp:=New object
$emp.firstName:="Mary"
$emp.lastName:="Smith"
$emp.employer:=New object("__KEY";121) //Existing PK in the related
dataClass Company
$emp.__NEW:=True
$empsCollection.push($emp)
$employees:=ds.Employee.fromCollection($empsCollection)

```

## Example 5

We want to create an entity. The \_\_NEW property is omitted, the employee primary key is given and does not exist:

```

var $empsCollection : Collection
var $emp : Object
var $employees : cs.EmployeeSelection

$empsCollection:=New collection
$emp:=New object
$emp.ID:=10000 //Unexisting primary key
$emp.firstName:="Françoise"
$emp.lastName:="Sagan"
$empsCollection.push($emp)
$employees:=ds.Employee.fromCollection($empsCollection)

```

## Example 6

In this example, the first entity will be created and saved but the second will fail since they both use the same primary key:

```

var $empsCollection : Collection
var $emp; $emp2 : Object
var $employees : cs.EmployeeSelection

$empsCollection:=New collection
$emp:=New object
$emp.ID:=10001 // Unexisting primary key
$emp.firstName:="Simone"
$emp.lastName:="Martin"
$emp.__NEW:=True
$empsCollection.push($emp)

$emp2:=New object
$emp2.ID:=10001 // Same primary key, already existing
$emp2.firstName:="Marc"
$emp2.lastName:="Smith"
$emp2.__NEW:=True
$empsCollection.push($emp2)
$employees:=ds.Employee.fromCollection($empsCollection)
//first entity is created
//duplicated key error for the second entity

```

## See also

[.toCollection\(\)](#)

## .get()

- History

`.get( primaryKey : Integer { ; settings : Object } ) : 4D.Entity`  
`.get( primaryKey : Text { ; settings : Object } ) : 4D.Entity`

| Parameter  | Type            | Description                                    |
|------------|-----------------|--|
| primaryKey | Integer OR Text | -> Primary key value of the entity to retrieve |
| settings   | Object          | -> Build option: context                       |
| Result     | 4D.Entity       | <- Entity matching the designated primary key  |

## Description

The `.get()` function queries the dataclass to retrieve the entity matching the *primaryKey* parameter.

In *primaryKey*, pass the primary key value of the entity to retrieve. The value type must match the primary key type set in the datastore (Integer or Text). You can also make sure that the primary key value is always returned as Text by using the [.getKey\(\)](#) function with the `dk key as string` parameter.

If no entity is found with *primaryKey*, a **Null** entity is returned.

Lazy loading is applied, which means that related data is loaded from disk only when it is required.

## settings

In the optional *settings* parameter, you can pass an object containing additional options. The following property is supported:

| Property | Type | Description   |
|----------|------|---|
| context  | Text | Label for the automatic optimization context applied to the entity. This context will be used by the subsequent code that loads the entity so that it can benefit from the optimization. This feature is <a href="#">designed for ORDA client/server processing</a> . |

## Example 1

```
var $entity : cs.EmployeeEntity
var $entity2 : cs.InvoiceEntity
$entity:=ds.Employee.get(167) // return the entity whose primary key
value is 167
$entity2:=ds.Invoice.get("DGGX20030") // return the entity whose
primary key value is "DGGX20030"
```

## Example 2

This example illustrates the use of the *context* property:

```

var $e1; $e2; $e3; $e4 : cs.EmployeeEntity
var $settings; $settings2 : Object

$settings:=New object("context";"detail")
$settings2:=New object("context";"summary")

$e1:=ds.Employee.get(1;$settings)
completeAllData($e1) // In completeAllData method, an optimization is
triggered and associated to context "detail"

$e2:=ds.Employee.get(2;$settings)
completeAllData($e2) // In completeAllData method, the optimization
associated to context "detail" is applied

$e3:=ds.Employee.get(3;$settings2)
completeSummary($e3) //In completeSummary method, an optimization is
triggered and associated to context "summary"

$e4:=ds.Employee.get(4;$settings2)
completeSummary($e4) //In completeSummary method, the optimization
associated to context "summary" is applied

```

## **.getCount()**

- History

**.getCount() : Integer**

| <b>Parameter</b> | <b>Type</b> | <b>Description</b>                     |
|------------------|-------------|--|
| result           | Integer<-   | Number of entities in the<br>dataclass |

### **Description**

The `.getCount()` function returns the number of entities in a dataclass.

If this function is used within a transaction, entities created during the transaction will be taken into account.

### **Example**

```

var $ds : 4D.DataStoreImplementation
var $number : Integer

$ds:=Open datastore(New object("hostname"; "www.myserver.com"); "myDS")

$number:=$ds.Persons.getCount()

```

## **.getDataStore()**

- History

**.getDataStore() : cs.DataStore**

| <b>Parameter</b> | <b>Type</b>    | <b>Description</b>            |
|------------------|----------------|-------------------------------|
| Result           | cs.DataStore<- | Datastore of the<br>dataclass |

### **Description**

The `.getDataStore()` function returns the datastore for the specified dataclass.

The datastore can be:

- the main datastore, as returned by the `ds` command.
- a remote datastore, opened using the `Open datastore` command.

## Example

The **SearchDuplicate** project method searches for duplicated values in any dataclass.

```
var $pet : cs.CatsEntity
$pet:=ds.Cats.all().first() //get an entity
SearchDuplicate($pet;"Dogs")
// SearchDuplicate method
// SearchDuplicate(entity_to_search;dataclass_name)

#DECLARE ($pet : Object ; $dataClassName : Text)
var $dataStore; $duplicates : Object

$dataStore:=$pet.getDataClass().getDataStore()
$duplicates:=$dataStore[$dataClassName].query("name=:1";$pet.name)
```

## .getInfo()

- History

### .getInfo() : Object

| Parameter | Type      | Description                  |
|-----------|-----------|------------------------------|
| Result    | Object <- | Information on the dataclass |

### Description

The `.getInfo()` function returns an object providing information about the dataclass. This function is useful for setting up generic code.

### Returned object

| Property    | Type    | Description                              |
|-------------|---------|--|
| exposed     | Boolean | True if the dataclass is exposed in REST |
| name        | Text    | Name of the dataclass                    |
| primaryKey  | Text    | Name of the primary key of the dataclass |
| tableNumber | Integer | Internal 4D table number                 |

### Example 1

```
#DECLARE ($entity : Object)
var $status : Object

computeEmployeeNumber($entity) //do some actions on entity

$status:=$entity.save()
if($status.success)
    ALERT("Record updated in table
"+$entity.getDataClass().getInfo().name)
End if
```

## Example 2

```
var $settings : Object
var $es : cs.ClientsSelection

$settings:=New object
$settings.parameters:=New object("receivedIds";getIds())
$settings.attributes:=New
object("pk";ds.Clients.getInfo().primaryKey)
$es:=ds.Clients.query(":pk in :receivedIds";$settings)
```

## Example 3

```
var $pk : Text
var $dataClassAttribute : Object

$pk:=ds.Employee.getInfo().primaryKey
$dataClassAttribute:=ds.Employee[$pk] // If needed the attribute
matching the primary key is accessible
```

## See also

[DataClassAttribute.exposed](#)

## .getRemoteCache()

- History

**.getRemoteCache() : Object**

| Parameter | Type     | Description   |
|-----------|----------|---|
| result    | Object<- | Object describing the contents of the ORDA cache for the dataclass. |

**Advanced mode:** This function is intended for developers who need to customize ORDA default features for specific configurations. In most cases, you will not need to use it.

## Description

The `.getRemoteCache()` function returns an object that holds the contents of the ORDA cache for a dataclass..

Calling this function from a 4D single-user application returns `Null`.

The returned object has the following properties:

| Property   | Type       | Description   |
|------------|------------|---|
| maxEntries | Integer    | Maximum number of entries collection.                                     |
| stamp      | Integer    | Stamp of the cache.   |
| timeout    | Integer    | Time remaining before the new entries in the cache are marked as expired. |
| entries    | Collection | Contains an entry object for each entity in the cache.                    |

Each entry object in the `entries` collection has the following properties:

| Property | Type    | Description                       |
|----------|---------|-----------------------------------|
| data     | Object  | Object holding data on the entry. |
| expired  | Boolean | True if the entry has expired.    |
| key      | Text    | Primary key of the entity.        |

The `data` object in each entry contains the following properties:

| Property  | Type    | Description  |
|---|---------|--|
| <code>__KEY</code>  | String  | Primary key of the entity  |
| <code>__STAMP</code>  | Longint | Timestamp of the entity in the database                                  |
| <code>__TIMESTAMP</code>  | String  | Stamp of the entity in the database (format is YYYY-MM-DDTHH:MM:SS:ms:Z) |
| If there is data in the cache for a dataclass attribute, it                                       |         |  |
| <code>dataClassAttributeNameVariant</code> is returned in a property with the same type as in the |         |  |
| database.   |         |  |

Data concerning related entities is stored in the cache of the data object.

## Example

In the following example, `$ds.Persons.all()` loads the first entity with all its attributes. Then, the request optimization is triggered, so only `firstname` and `address.city` are loaded.

Note that `address.city` is loaded in the cache of the `Persons` dataclass.

Only the first entity of the `Address` dataclass is stored in the cache. It is loaded during the first iteration of the loop.

```
var $ds : 4D.DataStoreImplementation
var $persons : cs.PersonsSelection
var $p : cs.PersonsEntity
var $cachePersons; $cacheAddress : Object
var $text : Text

$ds:=Open datastore(New object("hostname"; "www.myserver.com"); "myDS")

$persons:=$ds.Persons.all()

$text:=""
For each ($p; $persons)
    $text:=$p.firstname+" lives in "+$p.address.city+" / "
End for each

$cachePersons:=$ds.Persons.getRemoteCache()
$cacheAddress:=$ds.Adress.getRemoteCache()
```

## See also

[.setRemoteCacheSettings\(\)](#)  
[.clearRemoteCache\(\)](#)

## .new()

- History

[.new\(\) : 4D.Entity](#)

| Parameter | Type         | Description                       |
|-----------|--------------|-----------------------------------|
| Result    | 4D.Entity <- | New entity matching the Dataclass |

## Description

The `.new()` function creates in memory and returns a new blank entity related to the Dataclass.

The entity object is created in memory and is not saved in the database until the `.save()` function is called. If the entity is deleted before being saved, it cannot be recovered.

**4D Server:** In client-server, if the primary key of the corresponding table is auto-incremented, it will be calculated when the entity is saved on the server.

All attributes of the entity are initialized with the **null** value.

Attributes can be initialized with default values if the **Map NULL values to blank values** option is selected at the 4D database structure level.

## Example

This example creates a new entity in the "Log" Dataclass and records information in the "info" attribute:

```
var $entity : cs.LogEntity
$entity:=ds.Log.new() //create a reference
$entity.info:="New entry" //store some information
$entity.save() //save the entity
```

## .newSelection()

- History

**.newSelection( { keepOrder : Integer } ) : 4D.EntitySelection**

| Parameter | Type                  | Description   |
|-----------|-----------------------|---|
| keepOrder | Integer               | <code>dk keep ordered</code> : creates an ordered entity<br><code>-&gt; selection,</code><br><code>dk non ordered</code> : creates an unordered entity<br><code>selection (default if omitted)</code> |
| Result    | 4D.EntitySelection <- | New blank entity selection related to the dataclass   |

## Description

The `.newSelection()` function creates a new, blank, non-shareable entity selection, related to the dataclass, in memory.

For information on non-shareable entity selections, please refer to [this section](#).

If you want to create an ordered entity selection, pass the `dk keep ordered` selector in the `keepOrder` parameter. By default if you omit this parameter, or if you pass the `dk non ordered` selector, the method creates an unordered entity selection.

Unordered entity selections are faster but you cannot rely on entity positions. For more information, please see [Ordered vs Unordered entity selections](#).

When created, the entity selection does not contain any entities (`mySelection.length` returns 0). This method lets you build entity selections gradually by making subsequent calls to the [add\(\)](#) function.

## Example

```
var $USelection; $OSelection : cs.EmployeeSelection  
$USelection:=ds.Employee.newSelection() //create an unordered empty  
entity selection  
$OSelection:=ds.Employee.newSelection(dk keep ordered) //create an  
ordered empty entity selection
```

## .query()

- History

**.query(** *queryString* : Text { ; ...*value* : any } { ; *querySettings* : Object } **) :**  
4D.EntitySelection  
**.query(** *formula* : Object { ; *querySettings* : Object } **) :** 4D.EntitySelection

| Parameter                  | Type                          | Description   |
|----------------------------|-------------------------------|---|
| <i>queryString</i>         | Text                          | -> Search criteria as string  |
| <i>formula</i>             | Object                        | -> Search criteria as formula object  |
| <i>value</i>               | any                           | -> Value(s) to use for indexed placeholder(s)   |
| <i>querySettingsObject</i> |                               | -> Query options: parameters, attributes, args, allowFormulas, context, queryPath, queryPlan<br>New entity selection made up of entities from |
| <i>Result</i>              | 4D.EntitySelection<-dataclass | meeting the search criteria specified in<br><i>queryString</i> or <i>formula</i>  |

## Description

The `.query()` function searches for entities that meet the search criteria specified in *queryString* or *formula* and (optionally) *value*(s), for all the entities in the dataclass, and returns a new object of type `EntitySelection` containing all the entities that are found. Lazy loading is applied.

If no matching entities are found, an empty `EntitySelection` is returned.

### queryString parameter

The *queryString* parameter uses the following syntax:

```
attributePath|formula comparator value  
{logicalOperator attributePath|formula comparator value}  
{order by attributePath {desc | asc}}
```

where:

- **attributePath:** path of attribute on which you want to execute the query. This parameter can be a simple name (for example "country") or any valid attribute path (for example "country.name"). In case of an attribute path whose type is `Collection`, [ ] notation is used to handle all the occurrences (for example "children[ ].age").

*You cannot use directly attributes whose name contains special characters such as ".", "[ ]", or "=", ">", "#...", because they will be incorrectly evaluated in the query string. If you need to query on such*

*attributes, you must consider using placeholders, which allow an extended range of characters in attribute paths* (see **Using placeholders below**).

- **formula:** a valid formula passed as `Text` or `Object`. The formula will be evaluated for each processed entity and must return a boolean value. Within the formula, the entity is available through the `This` object.

- **Text:** the formula string must be preceded by the `eval()` statement, so that the query parser evaluates the expression correctly. For example: `"eval(length(This.lastname) >=30)"`
- **Object:** the [formula object](#) is passed as a **placeholder** (see below). The formula must have been created using the [Formula](#) or [Formula from string](#) command.
  - Keep in mind that 4D formulas only support `&` and `|` symbols as logical operators.
  - If the formula is not the only search criteria, the query engine optimizer could prior process other criteria (e.g. indexed attributes) and thus, the formula could be evaluated for only a subset of entities.

Formulas in queries can receive parameters through `$1`. This point is detailed in the **formula parameter** paragraph below.

- You can also pass directly a `formula` parameter object instead of the `queryString` parameter (recommended when formulas are more complex). See **formula parameter** paragraph below.
- For security reasons, formula calls within `query()` functions can be disallowed. See `querySettings` parameter description.

- **comparator:** symbol that compares `attributePath` and `value`. The following symbols are supported:

| Comparison                           | Symbol(s)      | Comment  |
|--------------------------------------|----------------|--|
| Equal to                             | =, ==          | Gets matching data, supports the wildcard (@), neither case-sensitive nor diacritic.                           |
|                                      | ==:, IS        | Gets matching data, considers the @ as a standard character, neither case-sensitive nor diacritic              |
| Not equal to                         | #, !=          | Supports the wildcard (@)  |
|                                      | !=:, IS<br>NOT | Considers the @ as a standard character  |
| Less than                            | <              |  |
| Greater than                         | >              |  |
| Less than or equal to                | <=             |  |
| Greater than or equal to             | >=             |  |
| Included in                          | IN             | Gets data equal to at least one of the values in a collection or in a set of values, supports the wildcard (@) |
| Not condition applied on a statement | NOT            | Parenthesis are mandatory when NOT is used before a statement containing several operators                     |
| Contains keyword                     | %              | Keywords can be used in attributes of string or picture type   |

- **value**: the value to compare to the current value of the property of each entity in the entity selection or element in the collection. It can be a **placeholder** (see **Using placeholders** below) or any expression matching the data type property. When using a constant value, the following rules must be respected:

- **text** type constant can be passed with or without simple quotes (see **Using quotes** below). To query a string within a string (a "contains" query), use the wildcard symbol (@) in value to isolate the string to be searched for as shown in this example: "@Smith@". The following keywords are forbidden for text constants: true, false.
- **boolean** type constants: true or false (case sensitive).
- **numeric** type constants: decimals are separated by a '.' (period).
- **date** type constants: "YYYY-MM-DD" format
- **null** constant: using the "null" keyword will find null and **undefined** properties.
- in case of a query with an IN comparator, *value* must be a collection, or values matching the type of the attribute path between [ ] separated by commas (for strings, " " characters must be escaped with \ ).

- **logicalOperator**: used to join multiple conditions in the query (optional). You can use one of the following logical operators (either the name or the symbol can be used):

### **Conjunction Symbol(s)**

AND           &, &&, and  
OR            ||, or

- **order by attributePath**: you can include an order by *attributePath* statement in the query so that the resulting data will be sorted according to that statement. You can use multiple order by statements, separated by commas (e.g., order by *attributePath1* desc, *attributePath2* asc). By default, the order is ascending. Pass 'desc' to define a descending order and 'asc' to define an ascending order.

If you use this statement, the returned entity selection is ordered (for more information, please refer to [Ordered vs Unordered entity selections](#)).

### **Using quotes**

When you use quotes within queries, you must use single quotes ' ' inside the query and double quotes " " to enclose the whole query, otherwise an error is returned. For example:

```
"employee.name = 'smith' AND employee.firstname = 'john'"
```

Single quotes ('') are not supported in searched values since they would break the query string. For example "comp.name = 'John's pizza' " will generate an error. If you need to search on values with single quotes, you may consider using placeholders (see below).

### **Using parenthesis**

You can use parentheses in the query to give priority to the calculation. For example, you can organize a query as follows:

```
"(employee.age >= 30 OR employee.age <= 65) AND (employee.salary <= 10000 OR employee.status = 'Manager')"
```

### **Using placeholders**

4D allows you to use placeholders for *attributePath*, *formula* and *value* arguments within the *queryString* parameter. A placeholder is a parameter that you insert in query strings and that is replaced by another value when the query string is evaluated. The value of placeholders is evaluated once at the beginning of the query; it is not evaluated for each element.

Two types of placeholders can be used: **indexed placeholders** and **named placeholders**:

### Indexed placeholders

Parameters are inserted as `:paramIndex` (for example `:1, :2...`) in *queryString* and their corresponding values are provided by the sequence of *value* parameter(s). You can use up to 128 *value* parameters

Example `$r:=class.query(":1=:2";"city";"Chicago")`

### Named placeholders

Parameters are inserted as `:paramName` (for example `:name`) and their values are provided by attributes and/or parameters of the *querySettings* parameter

```
$o.attributes:=New  
object("att";"city")  
$o.parameters:=New  
object("name";"Chicago")  
$r:=class.query(":att=
```

You can mix all argument kinds in *queryString*. A *queryString* can contain, for *attributePath*, *formula* and *value* parameters:

- direct values (no placeholders),
- indexed placeholders and/or named placeholders.

Using placeholders in queries **is recommended** for the following reasons:

1. It prevents malicious code insertion: if you directly use user-filled variables within the query string, a user could modify the query conditions by entering additional query arguments. For example, imagine a query string like:

```
$vquery:="status = 'public' & name = "+myname //user enters their  
name  
$result:=$col.query($vquery)
```

This query seems secured since non-public data are filtered. However, if the user enters in the *myname* area something like "*smith OR status='private'*", the query string would be modified at the interpretation step and could return private data.

When using placeholders, overriding security conditions is not possible:

```
$result:=$col.query("status='public' & name=:1";myname)
```

In this case if the user enters *smith OR status='private'* in the *myname* area, it will not be interpreted in the query string, but only passed as a value. Looking for a person named "smith OR status='private'" will just fail.

2. It prevents having to worry about formatting or character issues, especially when handling *attributePath* or *value* parameters that might contain non-alphanumeric characters such as ".", "["...
3. It allows the use of variables or expressions in query arguments. Examples:

```
$result:=$col.query("address.city = :1 & name  
=:2";$city;$myVar+"@")  
$result2:=$col.query("company.name = :1";"John's Pizzas")
```

## Looking for null values

When you look for null values, you cannot use the placeholder syntax because the query engine considers null as an unexpected comparison value. For example, if you execute the following query:

```
$vSingles:=ds.Person.query("spouse = :1";Null) // will NOT work
```

You will not get the expected result because the null value will be evaluated by 4D as an error resulting from the parameter evaluation (for example, an attribute coming from another query). For these kinds of queries, you must use the direct query syntax:

```
$vSingles:=ds.Person.query("spouse = null") //correct syntax
```

## Linking collection attribute query arguments



This feature is only available in queries on dataclasses and [entity selections](#). It cannot be used in queries on [collections](#).

When searching within dataclass object attributes containing collections using multiple query arguments joined by the AND operator, you may want to make sure that only entities containing elements that match all arguments are returned, and not entities where arguments can be found in different elements. To do this, you need to link query arguments to collection elements, so that only single elements containing linked arguments are found.

For example, with the following two entities:

```
Entity 1:  
ds.People.name: "martin"  
ds.People.places:  
  { "locations" : [ {  
      "kind":"home",  
      "city":"paris"  
    } ] }
```

```
Entity 2:  
ds.People.name: "smith"  
ds.People.places:  
  { "locations" : [ {  
      "kind":"home",  
      "city":"lyon"  
    } , {  
      "kind":"office",  
      "city":"paris"  
    } ] }
```

You want to find people with a "home" location kind in the city "paris". If you write:

```
ds.People.query("places.locations[].kind= :1 and  
places.locations[].city= :2";"home";"paris")
```

... the query will return "martin" **and** "smith" because "smith" has a "locations" element whose "kind" is "home" and a "locations" element whose "city" is "paris", even though they are different elements.

If you want to only get entities where matching arguments are in the same collection element, you need to **link arguments**. To link query arguments:

- Add a letter between the [] in the first path to link and repeat the same letter in all linked arguments. For example: `locations[a].city` and `locations[a].kind`. You can use any letter of the Latin alphabet (not case sensitive).
- To add different linked criteria in the same query, use another letter. You can create up to 26 combinations of criteria in a single query.

With the above entities, if you write:

```
ds.People.query("places.locations[a].kind= :1 and
places.locations[a].city= :2";"home";"paris")
```

... the query will only return "martin" because it has a "locations" element whose "kind" is "home" and whose "city" is "paris". The query will not return "smith" because the values "home" and "paris" are not in the same collection element.

## Queries in many-to-many relations

ORDA offers a special syntax to facilitate queries in many-to-many relations. In this context, you may need to search for different values with an `AND` operator BUT in the same attribute. For example, take a look at the following structure:

Imagine that you want to search all movies in which *both* actor A and actor B have a role. If you write a simple query using an `AND` operator, it will not work:

```
// invalid code
$es:=ds.Movie.query("roles.actor.lastName = :1 AND roles.actor.lastName
= :2";"Hanks";"Ryan")
// $es is empty
```

Basically, the issue is related to the internal logic of the query: you cannot search for an attribute whose value would be both "A" and "B".

To make it possible to perform such queries, ORDA allows a special syntax: you just need to add a *class index* between `{}` in all additional relation attributes used in the string:

```
"relationAttribute.attribute = :1 AND relationAttribute{x}.attribute =
:2 [AND relationAttribute{y}.attribute...]"
```

`{x}` tells ORDA to create another reference for the relation attribute. It will then perform all the necessary bitmap operations internally. Note that **x** can be any number **except 0**: {1}, or {2}, or {1540}... ORDA only needs a unique reference in the query for each class index.

In our example, it would be:

```
// valid code
$es:=ds.Movie.query("roles.actor.lastName = :1 AND
roles.actor{2}.lastName = :2";"Hanks";"Ryan")
// $es contains movies (You've Got Mail, Sleepless in Seattle, Joe
Versus the Volcano)
```

## formula parameter

As an alternative to formula insertion within the `queryString` parameter (see above), you can pass directly a formula object as a boolean search criteria. Using a formula object for queries is **recommended** since you benefit from tokenization, and code is easier to search/read.

The formula must have been created using the [Formula](#) or [Formula from string](#) command. In this case:

- the *formula* is evaluated for each entity and must return true or false. During the execution of the query, if the formula's result is not a boolean, it is considered as false.
- within the *formula*, the entity is available through the `This` object.
- if the `Formula` object is **null**, the error 1626 ("Expecting a text or formula") is generated, that you can intercept using a method installed with `ON ERR CALL`.

For security reasons, formula calls within `query()` functions can be disallowed. See `querySettings` parameter description.

## Passing parameters to formulas

Any *formula* called by the `query()` class function can receive parameters:

- Parameters must be passed through the **args** property (object) of the `querySettings` parameter.
- The formula receives this **args** object as a **\$1** parameter.

This small code shows the principles of how parameter are passed to methods:

```
$settings:=New object("args";New object("exclude";"-")) //args object  
to pass parameters  
$es:=ds.Students.query("eval(checkName($1.exclude))";$settings) //args  
is received in $1
```

Additional examples are provided in example 3.

**4D Server:** In client/server, formulas are executed on the server. In this context, only the `querySettings.args` object is sent to the formulas.

## querySettings parameter

In the `querySettings` parameter, you can pass an object containing additional options. The following properties are supported:

| Property   | Type   | Description   |
|------------|--------|---|
| parameters | Object | <b>Named placeholders for values</b> used in the <code>queryString</code> or <code>formula</code> . Values are expressed as property / value pairs, where property is the placeholder name inserted for a value in the <code>queryString</code> or <code>formula</code> ("":placeholder") and value is the value to compare. You can mix indexed placeholders (values directly passed in value parameters) and named placeholder values in the same query.<br><b>Named placeholders for attribute paths</b> used in the <code>queryString</code> or <code>formula</code> . Attributes are expressed as property / value pairs, where property is the placeholder name inserted for an attribute path in the <code>queryString</code> or <code>formula</code> ("":placeholder"), and value can be a string or a collection of strings. Each value is a path that can designate either a scalar or a related attribute of the dataclass or a property in an object field of the dataclass |
| attributes | Object | <b>Type</b><br>String<br><b>Description</b><br>attributePath expressed using the dot notation, e.g. "name" or "user.address.zipCode"  |

| Property      | Type                  | Description   |
|---------------|-----------------------|---|
|               | Collection of strings | Each string of the collection represents a level of attributePath, e.g. ["name"] or ["user","address","zipCode"]. Using a collection allows querying on attributes with names that are not compliant with dot notation, e.g. ["4Dv17.1","en/fr"]  |
| args          | Object                | You can mix indexed placeholders (values directly passed in value parameters) and named placeholder values in the same query.   |
| allowFormulas | Boolean               | Parameter(s) to pass to formulas, if any. The <b>args</b> object will be received in \$1 within formulas and thus its values will be available through <code>\$1.property</code> (see example 3).   |
| context       | Text                  | True to allow the formula calls in the query (default). Pass false to disallow formula execution. If set to false and <code>query()</code> is given a formula, an error is sent (1278 - Formula not allowed in this member method).   |
| queryPlan     | Boolean               | Label for the automatic optimization context applied to the entity selection. This context will be used by the code that handles the entity selection so that it can benefit from the optimization. This feature is designed for client/server processing; for more information, please refer to the <b>Client/server optimization</b> section.<br><br>In the resulting entity selection, returns or does not return the detailed description of the query just before it is executed, i.e. the planned query. The returned property is an object that includes each planned query and subquery (in the case of a complex query). This option is useful during the development phase of an application. It is usually used in conjunction with <code>queryPath</code> . Default if omitted: false. <b>Note:</b> This property is supported only by the <code>entitySelection.query()</code> and <code>dataClass.query()</code> functions. |
| queryPath     | Boolean               | In the resulting entity selection, returns or does not return the detailed description of the query as it is actually performed. The returned property is an object that contains the actual path used for the query (usually identical to that of the <code>queryPlan</code> , but may differ if the engine manages to optimize the query), as well as the processing time and the number of records found.<br><br>This option is useful during the development phase of an application. Default if omitted: false. <b>Note:</b> This property is supported only by the <code>entitySelection.query()</code> and <code>dataClass.query()</code> functions.   |

## About `queryPlan` and `queryPath`

The information recorded in `queryPlan`/`queryPath` include the query type (indexed and sequential) and each necessary subquery along with conjunction operators. Query paths also contain the number of entities found and the time required to execute each search criterion. You may find it useful to analyze this information while developing your application(s). Generally, the description of the query plan and its path are identical but they can differ because 4D can implement dynamic optimizations when a query is executed in order to improve performance. For example, the 4D engine can dynamically convert an indexed query into a sequential one if it estimates that it is faster. This particular case can occur when the number of entities being searched for is low.

For example, if you execute the following query:

```
$sel:=ds.Employee.query("salary < :1 and employer.name = :2 or
employer.revenues > :3";\
50000;"Lima West Kilo";10000000;New
object("queryPath";True;"queryPlan";True))
```

#### queryPlan:

```
{Or:[{And:[{item:[index : Employee.salary ] < 50000},
{item:Join on Table : Company : Employee.employerID = Company.ID,
subquery:[{item:[index : Company.name ] = Lima West Kilo}]}]}, {item:Join on Table : Company : Employee.employerID = Company.ID,
subquery:[{item:[index : Company.revenues ] > 10000000}]}]}
```

#### queryPath:

```
{steps:[{description:OR,time:63,recordsfounds:1388132,
steps:[{description:AND,time:32,recordsfounds:131,
steps:[{description:[index : Employee.salary ] <
50000,time:16,recordsfounds:728260},{description:Join on Table :
Company : Employee.employerID = Company.ID,time:0,recordsfounds:131,
steps:[{steps:[{description:[index : Company.name ] = Lima West
Kilo,time:0,recordsfounds:1}]}]}, {description:Join on Table : Company
: Employee.employerID = Company.ID,time:31,recordsfounds:1388132,
steps:[{steps:[{description:[index : Company.revenues ] >
10000000,time:0,recordsfounds:933}]}]}]}]}
```

## Example 1

This section provides various examples of queries.

#### Query on a string:

```
$entitySelection:=ds.Customer.query("firstName = 'S@'")
```

#### Query with a NOT statement:

```
$entitySelection:=ds.Employee.query("not (firstName=Kim) ")
```

#### Queries with dates:

```
$entitySelection:=ds.Employee.query("birthDate > :1";"1970-01-01")
$entitySelection:=ds.Employee.query("birthDate <= :1";Current date-
10950)
```

#### Query with indexed placeholders for values:

```
$entitySelection:=ds.Customer.query("(firstName = :1 or firstName = :2)
and (lastName = :3 or lastName = :4)";"D@";"R@";"S@";"K@")
```

#### Query with indexed placeholders for values on a related dataclass:

```
$entitySelection:=ds.Employee.query("lastName = :1 and manager.lastName
= :2";"M@";"S@")
```

#### Query with indexed placeholder including a descending order by statement:

```
$entitySelection:=ds.Student.query("nationality = :1 order by
campus.name desc, lastname";"French")
```

#### Query with named placeholders for values:

```

var $querySettings : Object
var $managedCustomers : cs.CustomerSelection
$querySettings:=New object
$querySettings.parameters:=New object("userId";1234;"extraInfo";New
object("name";"Smith"))
$managedCustomers:=ds.Customer.query("salesperson.userId = :userId and
name = :extraInfo.name";$querySettings)

```

**Query that uses both named and indexed placeholders for values:**

```

var $querySettings : Object
var $managedCustomers : cs.CustomerSelection
$querySettings.parameters:=New object("userId";1234)
$managedCustomers:=ds.Customer.query("salesperson.userId = :userId and
name=:1;"Smith";$querySettings)

```

**Query with queryPlan and queryPath objects:**

```

$entitySelection:=ds.Employee.query("(firstName = :1 or firstName = :2)
and (lastName = :3 or lastName = :4)";"D@";"R@";"S@";"K@";New
object("queryPlan";True;"queryPath";True))

```

```

//you can then get these properties in the resulting entity selection
var $queryPlan; $queryPath : Object
$queryPlan:=$entitySelection.queryPlan
$queryPath:=$entitySelection.queryPath

```

**Query with an attribute path of Collection type:**

```

$entitySelection:=ds.Employee.query("extraInfo.hobbies[] .name =
:1;"horsebackriding")

```

**Query with an attribute path of Collection type and linked attributes:**

```

$entitySelection:=ds.Employee.query("extraInfo.hobbies[a] .name = :1 and
extraInfo.hobbies[a] .level=:2;"horsebackriding";2)

```

**Query with an attribute path of Collection type and multiple linked attributes:**

```

$entitySelection:=ds.Employee.query("extraInfo.hobbies[a] .name = :1 and
extraInfo.hobbies[a] .level = :2 and extraInfo.hobbies[b] .name = :3 and
extraInfo.hobbies[b] .level = :4;"horsebackriding";2;"Tennis";5)

```

**Query with an attribute path of Object type:**

```

$entitySelection:=ds.Employee.query("extra.eyeColor = :1;"blue")

```

**Query with an IN statement:**

```

$entitySelection:=ds.Employee.query("firstName in :1";New
collection("Kim";"Dixie"))

```

**Query with a NOT (IN) statement:**

```

$entitySelection:=ds.Employee.query("not (firstName in :1)";New
collection("John";"Jane"))

```

**Query with indexed placeholders for attributes:**

```

var $es : cs.EmployeeSelection
$es:=ds.Employee.query(":1 = 1234 and :2 =
'Smith';"salesperson.userId";"name")
//salesperson is a related entity

```

Query with indexed placeholders for attributes and named placeholders for values:

```
var $es : cs.EmployeeSelection
var $querySettings : Object
$querySettings:=New object
$querySettings.parameters:=New object ("customerName"; "Smith")
$es:=ds.Customer.query(":1 = 1234 and :2 =
:customerName"; "salesperson.userId"; "name"; $querySettings)
//salesperson is a related entity
```

Query with indexed placeholders for attributes and values:

```
var $es : cs.EmployeeSelection
$es:=ds.Clients.query(":1 = 1234 and :2 =
:3"; "salesperson.userId"; "name"; "Smith")
//salesperson is a related entity
```

## Example 2

This section illustrates queries with named placeholders for attributes.

Given an Employee dataclass with 2 entities:

Entity 1:

```
name: "Marie"
number: 46
softwares:{
"Word 10.2": "Installed",
"Excel 11.3": "To be upgraded",
"Powerpoint 12.4": "Not installed"
}
```

Entity 2:

```
name: "Sophie"
number: 47
softwares:{
"Word 10.2": "Not installed",
"Excel 11.3": "To be upgraded",
"Powerpoint 12.4": "Not installed"
}
```

Query with named placeholders for attributes:

```
var $querySettings : Object
var $es : cs.EmployeeSelection
$querySettings:=New object
$querySettings.attributes:=New object ("attName"; "name"; "attWord"; New
collection("softwares"; "Word 10.2"))
$es:=ds.Employee.query(":attName = 'Marie' and :attWord =
'Installed"'; $querySettings)
//$es.length=1 (Employee Marie)
```

Query with named placeholders for attributes and values:

```

var $querySettings : Object
var $es : cs.EmployeeSelection
var $name : Text
$querySettings:=New object
  //Named placeholders for values
  //The user is asked for a name
$name:=Request("Please enter the name to search:")
If (OK=1)
  $querySettings.parameters:=New object("givenName";$name)
  //Named placeholders for attribute paths
  $querySettings.attributes:=New object("attName";"name")
  $es:=ds.Employee.query(":attName= :givenName";$querySettings)
End if

```

### Example 3

These examples illustrate the various ways to use formulas with or without parameters in your queries.

The formula is given as text with `eval()` in the *queryString* parameter:

```

var $es : cs.StudentsSelection
$es:=ds.Students.query("eval(length(This.lastname) >=30) and
nationality='French'")

```

The formula is given as a `Formula` object through a placeholder:

```

var $es : cs.StudentsSelection
var $formula : Object
$formula:=Formula(Length(This.lastname)>=30)
$es:=ds.Students.query(":1 and nationality='French'";$formula)

```

Only a `Formula` object is given as criteria:

```

var $es : cs.StudentsSelection
var $formula : Object
$formula:=Formula(Length(This.lastname)>=30)
$es:=ds.Students.query($formula)

```

Several formulas can be applied:

```

var $formula1; $1; $formula2 ;$0 : Object
$formula1:=$1
$formula2:=Formula(Length(This.firstname)>=30)
$0:=ds.Students.query(":1 and :2 and
nationality='French'";$formula1;$formula2)

```

A text formula in *queryString* receives a parameter:

```

var $es : cs.StudentsSelection
var $settings : Object
$settings:=New object()
$settings.args:=New object("filter";"-")
$es:=ds.Students.query("eval(checkName($1.filter)) and
nationality=:1";"French";$settings)
  //checkName method
#DECLARE($exclude : Text) -> $result : Boolean
$result:=(Position($exclude;This.lastname)=0)

```

Using the same **checkName** method, a `Formula` object as placeholder receives a parameter:

```

var $es : cs.StudentsSelection
var $settings; $formula : Object
$formula:=Formula(checkName($1.filter))
$settings:=New object()
$settings.args:=New object("filter";"-")
$es:=ds.Students.query(":1 and
nationality=:2";$formula;"French";$settings)
$settings.args.filter:="*" // change the parameters without updating
the $formula object
$es:=ds.Students.query(":1 and
nationality=:2";$formula;"French";$settings)

```

We want to disallow formulas, for example when the user enters their query:

```

var $es : cs.StudentsSelection
var $settings : Object
var $queryString : Text
$queryString:=Request("Enter your query:")
if(OK=1)
    $settings:=New object("allowFormulas";False)
    $es:=ds.Students.query($queryString;$settings) //An error is raised
if $queryString contains a formula
End if

```

## See also

[.query\(\)](#) for entity selections

## .setRemoteCacheSettings()

- History

**.setRemoteCacheSettings(*settings* : Object)**

| Parameter Type  | Description  |
|-----------------|--|
| settings Object | - Object that sets the timeout and maximum size of the ORDA > cache for the dataclass. |

**Advanced mode:** This function is intended for developers who need to customize ORDA default features for specific configurations. In most cases, you will not need to use it.

### Description

The `.setRemoteCacheSettings()` function sets the timeout and maximum size of the ORDA cache for a dataclass..

In the *settings* parameter, pass an object with the following properties:

| Property Type     | Description                 |
|-------------------|-----------------------------|
| timeout Integer   | Timeout in seconds.         |
| maxEntriesInteger | Maximum number of entities. |

`timeout` sets the timeout of the ORDA cache for the dataclass (default is 30 seconds). Once the timeout has passed, the entities of the dataclass in the cache are considered as expired. This means that:

- the data is still there
- the next time the data is needed, it will be asked to the server
- ORDA automatically removes expired data when the maximum number of entities is

reached

Setting a `timeout` property sets a new timeout for the entities already present in the cache. It is useful when working with data that does not change very frequently, and thus when new requests to the server are not necessary.

`maxEntries` sets the max number of entities in the ORDA cache. Default is 30 000.

The minimum number of entries is 300, so the value of `maxEntries` must be equal to or higher than 300. Otherwise it is ignored and the maximum number of entries is set to 300.

If no valid properties are passed as `timeout` and `maxEntries`, the cache remains unchanged, with its default or previously set values.

When an entity is saved, it is updated in the cache and expires once the timeout is reached.

## Example

```
var $ds : 4D.DataStoreImplementation  
$ds:=Open datastore(New object("hostname"; "www.myserver.com"); "myDS")  
$ds.Buildings.setRemoteCacheSettings(New object("timeout"; 60;  
"maxEntries"; 350))
```

## See also

[.clearRemoteCache\(\)](#)  
[.getRemoteCache\(\)](#)

## DataClassAttribute

Dataclass attributes are available as properties of their respective classes. For example:

```
nameAttribute:=ds.Company.name //reference to class attribute  
revenuesAttribute:=ds.Company["revenues"] //alternate way
```

This code assigns to *nameAttribute* and *revenuesAttribute* references to the name and revenues attributes of the Company class. This syntax does NOT return values held inside of the attribute, but instead returns references to the attributes themselves. To handle values, you need to go through [Entities](#).

`DataClassAttribute` objects have properties that you can read to get information about your dataclass attributes.

Dataclass attribute objects can be modified, but the underlying database structure will not be altered.

### Summary

[\*\*.autoFilled : Boolean\*\*](#) contains True if the attribute value is automatically filled by 4D

[\*\*.exposed : Boolean\*\*](#) true if the attribute is exposed in REST

[\*\*.fieldNumber : Integer\*\*](#) contains the internal 4D field number of the attribute

[\*\*.FieldType : Integer\*\*](#) contains the 4D database type of the attribute

[\*\*.indexed : Boolean\*\*](#) contains **True** if there is a B-tree or a Cluster B-tree index on the attribute

[\*\*.inverseName : Text\*\*](#) returns the name of the attribute which is at the other side of the relation

[\*\*.keywordIndexed : Boolean\*\*](#) contains **True** if there is a keyword index on the attribute

[\*\*.kind : Text\*\*](#) returns the category of the attribute

[\*\*.mandatory : Boolean\*\*](#) contains True if Null value input is rejected for the attribute

[\*\*.name : Text\*\*](#) returns the name of the `dataClassAttribute` object as string

[\*\*.path : Text\*\*](#) returns the path of an alias attribute based upon a relation

[\*\*.readOnly : Boolean\*\*](#) true if the attribute is read-only

[\*\*.relatedDataClass : Text\*\*](#) returns the name of the dataclass related to the attribute

[\*\*.type : Text\*\*](#) contains the conceptual value type of the attribute

[\*\*.unique : Boolean\*\*](#) contains True if the attribute value must be unique

### **.autoFilled**

- History

**.autoFilled : Boolean**

### Description

The `.autoFilled` property contains True if the attribute value is automatically filled by 4D. This property corresponds to the following 4D field properties:

- "Autoincrement", for numeric type fields
- "Auto UUID", for UUID (alpha type) fields.

This property is not returned if `.kind` = "relatedEntity" or "relatedEntities".

For generic programming, you can use **Bool**(dataClassAttribute.autoFilled) to get a valid value (false) even if `.autoFilled` is not returned.

## .exposed

- History

**.exposed** : Boolean

### Description

The `.exposed` property is true if the attribute is exposed in REST.

### See also

[DataClass.getInfo\(\)](#)

## .fieldNumber

- History

**.fieldNumber** : Integer

### Description

The `.fieldNumber` property contains the internal 4D field number of the attribute.

This property is not returned if `.kind` = "relatedEntity" or "relatedEntities".

For generic programming, you can use **Num**(dataClassAttribute.fieldNumber) to get a valid value (0) even if `.fieldNumber` is not returned.

## .fieldType

- History

**.fieldType** : Integer

### Description

The `.fieldType` property contains the 4D database type of the attribute. It depends on the attribute kind (see [.kind](#)).

### Possible values:

| <b>dataClassAttribute.kind</b> | <b>fieldType</b>  |
|--------------------------------|---|
| storage                        | Corresponding 4D field type, see <a href="#">Value type</a>   |
| relatedEntity                  | 38 ( <code>Is object</code> )   |
| relatedEntities                | 42 ( <code>Is collection</code> ) <ul style="list-style-type: none"><li>• scalar: corresponding 4D field type, see <a href="#">Value type</a></li><li>• entity: 38 (<code>Is object</code>)</li><li>• entity selection: 42 (<code>Is collection</code>)</li></ul> |
| calculated                     | <ul style="list-style-type: none"><li>• scalar: corresponding 4D field type, see <a href="#">Value type</a></li><li>• entity: 38 (<code>Is object</code>)</li><li>• entity selection: 42 (<code>Is collection</code>)</li></ul>                                   |
| alias                          | <ul style="list-style-type: none"><li>• scalar: corresponding 4D field type, see <a href="#">Value type</a></li><li>• entity: 38 (<code>Is object</code>)</li><li>• entity selection: 42 (<code>Is collection</code>)</li></ul>                                   |

## See also

[.type](#)

## .indexed

- History

**.indexed** : Boolean

### Description

The `.indexed` property contains **True** if there is a B-tree or a Cluster B-tree index on the attribute.

This property is not returned if `.kind` = "relatedEntity" or "relatedEntities".

For generic programming, you can use **Bool**(dataClassAttribute.indexed) to get a valid value (false) even if `.indexed` is not returned.

## .inverseName

- History

**.inverseName** : Text

### Description

The `.inverseName` property returns the name of the attribute which is at the other side of the relation.

This property is not returned if `.kind` = "storage". It must be of the "relatedEntity" or "relatedEntities" kind.

For generic programming, you can use **String**(dataClassAttribute.inverseName) to get a valid value ("") even if `.inverseName` is not returned.

## .keywordIndexed

- History

**.keywordIndexed** : Boolean

### Description

The `.keywordIndexed` property contains **True** if there is a keyword index on the attribute.

This property is not returned if `.kind` = "relatedEntity" or "relatedEntities".

For generic programming, you can use **Bool**(dataClassAttribute.keywordIndexed) to get a valid value (false) even if `.keywordIndexed` is not returned.

## .kind

- History

.kind : Text

## Description

The `.kind` property returns the category of the attribute. Returned value can be one of the following:

- "storage": storage (or scalar) attribute, i.e. attribute storing a value, not a reference to another attribute
- "calculated": computed attribute, i.e. defined through a [get function](#)
- "alias": attribute built upon [another attribute](#)
- "relatedEntity": N -> 1 relation attribute (reference to an entity)
- "relatedEntities": 1 -> N relation attribute (reference to an entity selection)

## Example

Given the following table and relation:

The screenshot shows the DataArchitect interface. On the left, there is a table named 'Employee' with the following columns and data:

| ID         | 2 <sup>32</sup> |
|------------|-----------------|
| firstname  | A               |
| lastname   | A               |
| salary     | 0.5             |
| birthdate  | 17              |
| woman      | ■               |
| managerID  | 2 <sup>32</sup> |
| employerID | 2 <sup>32</sup> |

On the right, there is an 'Inspector' window titled 'Relation' for 'Table N°1 -> Table N°1'. The 'Many to One Options' section is expanded, showing:

- Name: manager
- Manual dropdown set to 'Manual'
- Auto Wildcard support
- Wildcard Choice dropdown with options: ID, firstname, lastname
- Prompt if related one does not exist
- One to Many Options section is collapsed
- Deletion Control and SQL sections are collapsed

```
var $attKind : Text
$attKind:=ds.Employee.lastname.kind // $attKind="storage"
$attKind:=ds.Employee.manager.kind // $attKind="relatedEntity"
$attKind:=ds.Employee.directReports.kind // $attKind="relatedEntities"
```

## .mandatory

- History

.mandatory : Boolean

## Description

The `.mandatory` property contains True if Null value input is rejected for the attribute.

This property is not returned if `.kind` = "relatedEntity" or "relatedEntities".

For generic programming, you can use `Bool(dataClassAttribute.mandatory)` to get a valid value (false) even if `.mandatory` is not returned.

**Warning:** This property corresponds to the "Reject NULL value input" field property at the 4D database level. It is unrelated to the existing "Mandatory" property which is a data entry control option for a table.

## .name

- History

**.name** : Text

### Description

The `.name` property returns the name of the `dataClassAttribute` object as string.

### Example

```
var $attName : Text  
$attName:=ds.Employee.lastname.name // $attName="lastname"
```

## .path

- History

**.path** : Text

### Description

The `.path` property returns the path of an alias attribute based upon a relation.

### Example

```
var $path : Text  
$path:=ds.Teacher.students.path // $path="courses.student"
```

## .readOnly

- History

**.readOnly** : Boolean

### Description

The `.readOnly` property is true if the attribute is read-only.

For example, computed attributes without [set function](#) are read-only.

## .relatedDataClass

- History

**.relatedDataClass** : Text

### Description

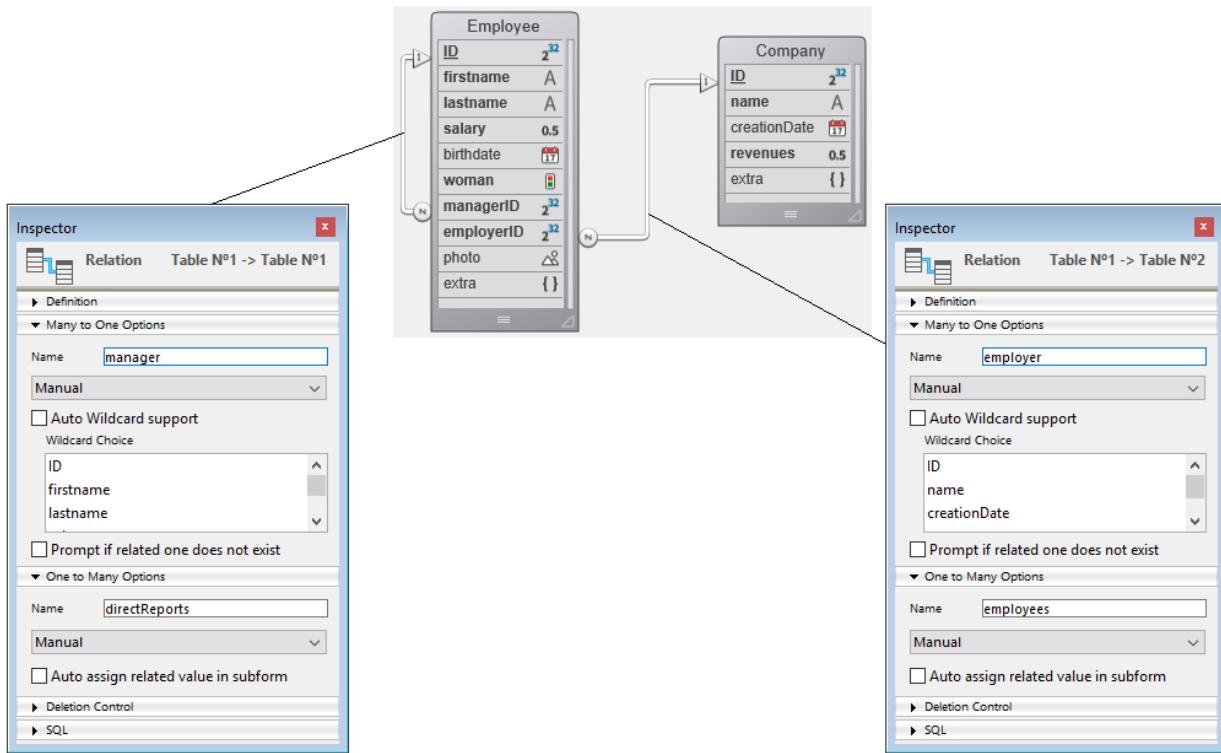
This property is only available with attributes of the "relatedEntity" or "relatedEntities" [.kind](#) property.

The `.relatedDataClass` property returns the name of the dataclass related to the

attribute.

## Example

Given the following tables and relations:



```
var $relClass1; $relClassN : Text
$relClass1:=ds.Employee.employer.relatedDataClass
// $relClass1="Company"
$relClassN:=ds.Employee.directReports.relatedDataClass
// $relClassN="Employee"
```

## .type

- History

### .type : Text

#### Description

The `.type` property contains the conceptual value type of the attribute, useful for generic programming.

The conceptual value type depends on the attribute [.kind](#).

#### Possible values:

| <code>dataClassAttribute.kind</code> | <code>type</code>   | <code>Comment</code>   |
|--------------------------------------|---|--|
| storage                              | "blob", "bool",<br>"date",<br>"image",<br>"number",<br>"object", or<br>"string"   | "number" is returned for any numeric types including duration. "string" is returned for uuid, alpha and text field types. "blob" attributes are <a href="#">blob objects</a> , they are handled using the <a href="#">Blob class</a> . |
| relatedEntity                        | related<br>dataClass<br>name  | Ex: "Companies"  |
| relatedEntities                      | related<br>dataClass<br>name +<br>"Selection"<br>suffix<br>• storage:<br>type ("blob",<br>"number",<br>etc.)<br>• entity:<br>selection:<br>dataClass<br>name +<br>"Selection" | Ex: "EmployeeSelection"  |
| calculated                           | dataClass<br>name<br>• entity<br>selection:<br>dataClass<br>name +<br>"Selection"   |  |

## See also

[.fieldType](#)

## .unique

- History

**.unique** : Boolean

## Description

The `.unique` property contains True if the attribute value must be unique. This property corresponds to the "Unique" 4D field property.

This property is not returned if `.kind` = "relatedEntity" or "relatedEntities".

For generic programming, you can use `Bool(dataClassAttribute.unique)` to get a valid value (false) even if `.unique` is not returned.

## DataStore

A [Datastore](#) is the interface object provided by ORDA to reference and access a database. `Datastore` objects are returned by the following commands:

- [`ds`](#): a shortcut to the main datastore
- [`Open datastore`](#): to open any remote datastore

## Summary

**.cancelTransaction()** cancels the transaction

**.clearAllRemoteContexts()** clears all the attributes for all the active contexts in the datastore

**.dataclassName : 4D.DataClass** contains a description of the dataclass

**.encryptionStatus(): Object** returns an object providing the encryption status for the current data file

**.flushAndLock()** flushes the cache of the local datastore and prevents other processes from performing write operations on the database

**.getAllRemoteContexts() : Collection** returns a collection of objects containing information on all the active optimization contexts in the datastore

**.getInfo(): Object** returns an object providing information about the datastore

**.getRemoteContextInfo(contextName : Text) : Object** returns an object that holds information on the *contextName* optimization context in the datastore.

**.getRequestLog() : Collection** returns the ORDA requests logged in memory on the client side

**.locked() : Boolean** returns True if the local datastore is currently locked

**.makeSelectionsAlterable()** sets all entity selections as alterable by default in the current application datastores

**.provideDataKey( curPassPhrase : Text ) : Object**

**.provideDataKey( curDataKey : Object ) : Object** allows providing a data encryption key for the current data file of the datastore and detects if the key matches the encrypted data

**.setAdminProtection( status : Boolean )** allows disabling any data access on the [web admin port](#), including for the [Data Explorer](#) in [WebAdmin](#) sessions

**.setRemoteContextInfo( contextName : Text ; dataClassName : Text ; attributes : Text {; contextType : Text {; pageLength : Integer}} )**

**.setRemoteContextInfo( contextName : Text ; dataClassName : Text; attributesColl : Collection {; contextType : Text {; pageLength : Integer }} )**

**.setRemoteContextInfo( contextName : Text ; dataClassObject : 4D.DataClass ; attributes : Text {; contextType : Text {; pageLength : Integer }} )**

**.setRemoteContextInfo( contextName : Text ; dataClassObject : 4D.DataClass ; attributesColl : Collection {; contextType : Text {; pageLength : Integer }} )** links the specified dataclass attributes to the *contextName* optimization context

**.startRequestLog()**

**.startRequestLog( file : 4D.File )**

**.startRequestLog( file : 4D.File ; options : Integer )**

**.startRequestLog( reqNum : Integer )** starts the logging of ORDA requests on the client side or on the server side

**.startTransaction()** starts a transaction in the current process on the database matching the datastore to which it applies

**.stopRequestLog()** stops any logging of ORDA requests on the machine it is called (client or server)

**.unlock()** removes the current lock on write operations in the datastore, if it has been set in the same process

**.validateTransaction()** accepts the transaction

## ds

- History

**ds { ( localID : Text ) } : cs.DataStore**

| Parameter | Type         | Description                                   |
|-----------|--------------|---|
| localID   | Text         | -> Local ID of the remote datastore to return |
| Result    | cs.DataStore | <- Reference to the datastore                 |

## Description

The `ds` command returns a reference to the datastore matching the current 4D database or the database designated by *localID*.

If you omit the *localID* parameter (or pass an empty string ""), the command returns a reference to the datastore matching the local 4D database (or the 4D Server database in case of opening a remote database on 4D Server). The datastore is opened automatically and available directly through `ds`.

You can also get a reference on an open remote datastore by passing its local id in the *localID* parameter. The datastore must have been previously opened with the [Open datastore](#) command by the current database (host or component). The local id is defined when using this command.

The scope of the local id is the database where the datastore has been opened.

If no *localID* datastore is found, the command returns **Null**.

Objects available in the `cs.Datastore` are mapped from the target database with respect to the [ORDA general rules](#).

## Example 1

Using the main datastore on the 4D database:

```
$result:=ds.Employee.query("firstName = :1";"S@")
```

## Example 2

```
var $connectTo; $firstFrench; $firstForeign : Object
var $frenchStudents; $foreignStudents : cs.DataStore
$connectTo:=New object("type";"4D
Server";"hostname";"192.168.18.11:8044")
$frenchStudents:=Open datastore($connectTo;"french")

$connectTo.hostname:="192.168.18.11:8050"
$foreignStudents:=Open datastore($connectTo;"foreign")
//...
//...
$firstFrench:=getFirst("french";"Students")
$firstForeign:=getFirst("foreign";"Students")
//getFirst method
//getFirst(localID;dataclass) -> entity
#DECLARE( $localId : Text; $dataClassName : Text ) -> $entity :
4D.Entity

$0:=ds ($localId) [$dataClassName].all().first()
```

## Open datastore

- History

## **Open datastore( *connectionInfo* : Object ; *localID* : Text ) : cs.DataStore**

| Parameter            | Type           | Description  |
|----------------------|----------------|--|
| connectionInfoObject | -> Object      | Connection properties used to reach the remote datastore                     |
| localID              | Text           | -> Id to assign to the opened datastore on the local application (mandatory) |
| Result               | cs.DataStore<- | Datastore object   |

### **Description**

The `Open datastore` command connects the application to the 4D database identified by the *connectionInfo* parameter and returns a matching `cs.DataStore` object associated with the *localID* local alias.

The *connectionInfo* 4D database must be available as a remote datastore, i.e.:

- its web server must be launched with http and/or https enabled,
- its **Expose as REST server** option must be checked,
- at least one client license is available.

If no matching database is found, `Open datastore` returns **Null**.

*localID* is a local alias for the session opened on remote datastore. If *localID* already exists on the application, it is used. Otherwise, a new *localID* session is created when the datastore object is used.

Objects available in the `cs.Datastore` are mapped from the target database with respect to the [ORDA general rules](#).

Once the session is opened, the following statements become equivalent and return a reference on the same datastore object:

```
$myds:=Open datastore(connectionInfo;"myLocalId")
$myds2:=ds("myLocalId")
// $myds and $myds2 are equivalent
```

Pass in *connectionInfo* an object describing the remote datastore you want to connect to. It can contain the following properties (all properties are optional except *hostname*):

| Property    | Type    | Description  |
|-------------|---------|--|
| hostname    | Text    | Name or IP address of the remote database + ":" + port number (port number is mandatory)   |
| user        | Text    | User name  |
| password    | Text    | User password  |
| idleTimeout | Longint | Inactivity session timeout (in minutes), after which the session is automatically closed by 4D. If omitted, default value is 60 (1h). The value cannot be < 60 (if a lower value is passed, the timeout is set to 60). For more information, see <b>Closing sessions</b> . |
| tls         | Boolean | Use secured connection(*). If omitted, false by default. Using a secured connection is recommended whenever possible.  |
| type        | Text    | Must be "4D Server"  |

(\*) If *tls* is true, the HTTPS protocol is used if:

- HTTPS is enabled on the remote datastore
- the given port is the right HTTPS port configured in the database settings

- a valid certificate and private encryption key are installed in the database.  
Otherwise, error "1610 - A remote request to host xxx has failed" is raised

## Example 1

Connection to a remote datastore without user / password:

```
var $connectTo : Object
var $remoteDS : cs.DataStore
$connectTo:=New object("type";"4D
Server";"hostname";"192.168.18.11:8044")
$remoteDS:=Open datastore($connectTo;"students")
ALERT("This remote datastore contains
"+String($remoteDS.Students.all().length)+" students")
```

## Example 2

Connection to a remote datastore with user / password / timeout / tls:

```
var $connectTo : Object
var $remoteDS : cs.DataStore
$connectTo:=New object("type";"4D
Server";"hostname";"\\"192.168.18.11:4443"\";
"user";"marie";"password";$pwd;"idleTimeout";70;"tls";True)
$remoteDS:=Open datastore($connectTo;"students")
ALERT("This remote datastore contains
"+String($remoteDS.Students.all().length)+" students")
```

## Example 3

Working with several remote datastores:

```
var $connectTo : Object
var $frenchStudents; $foreignStudents : cs.DataStore
$connectTo:=New object("hostname";"192.168.18.11:8044")
$frenchStudents:=Open datastore($connectTo;"french")
$connectTo.hostname:="192.168.18.11:8050"
$foreignStudents:=Open datastore($connectTo;"foreign")
ALERT("They are "+String($frenchStudents.Students.all().length)+" French students")
ALERT("They are "+String($foreignStudents.Students.all().length)+" foreign students")
```

## Error management

In case of error, the command returns **Null**. If the remote datastore cannot be reached (wrong address, web server not started, http and https not enabled...), error 1610 "A remote request to host XXX has failed" is raised. You can intercept this error with a method installed by `ON ERR CALL`.

## **.dataclassName**

- History

**.dataclassName** : 4D.DataClass

## Description

Each dataclass in a datastore is available as a property of the [DataStore object](#) data. The returned object contains a description of the dataclass.

## Example

```
var $emp : cs.Employee  
var $sel : cs.EmployeeSelection  
$emp:=ds.Employee // $emp contains the Employee dataclass  
$sel:=$emp.all() // gets an entity selection of all employees  
  
// you could also write directly:  
$sel:=ds.Employee.all()
```

## .cancelTransaction()

- History

### .cancelTransaction()

| Parameter Type | Description                     |
|----------------|---------------------------------|
|                | Does not require any parameters |

### Description

The `.cancelTransaction()` function cancels the transaction opened by the [.startTransaction\(\)](#) function at the corresponding level in the current process for the specified datastore.

The `.cancelTransaction()` function cancels any changes made to the data during the transaction.

You can nest several transactions (sub-transactions). If the main transaction is cancelled, all of its sub-transactions are also cancelled, even if they were validated individually using the [.validateTransaction\(\)](#) function.

## Example

See example for the [.startTransaction\(\)](#) function.

## .clearAllRemoteContexts()

- History

### .clearAllRemoteContexts()

| Parameter Type | Description                     |
|----------------|---------------------------------|
|                | Does not require any parameters |

### Description

The `.clearAllRemoteContexts()` function clears all the attributes for all the active contexts in the datastore.

This function is mainly used in the context of debugging. One thing to keep in mind is that when you open the debugger, it sends requests to the server and queries all the dataclass attributes to display them. This can overload your contexts with unnecessary data.

In such cases, you can use `.clearAllRemoteContexts()` to clear your contexts and

keep them clean.

## See also

[.getRemoteContextInfo\(\)](#)  
[.getAllRemoteContexts\(\)](#)  
[.setRemoteContextInfo\(\)](#)

## .encryptionStatus()

- History

**.encryptionStatus(): Object**

| Parameter | Type      | Description   |
|-----------|-----------|---|
| Result    | Object <- | Information about the encryption of the current datastore and of each table |

### Description

The `.encryptionStatus()` function returns an object providing the encryption status for the current data file (i.e., the data file of the `ds` datastore). The status for each table is also provided.

Use the `Data file encryption status` command to determine the encryption status of any other data file.

### Returned value

The returned object contains the following properties:

| Property         | Type    | Description  |
|------------------|---------|--|
| isEncrypted      | Boolean | True if the data file is encrypted   |
| keyProvided      | Boolean | True if the encryption key matching the encrypted data file is provided(*).        |
| tables           | Object  | Object containing as many properties as there are encryptable or encrypted tables. |
| <i>tableName</i> | Object  | Encryptable or Encrypted table   |
| name             | Text    | Name of the table  |
| num              | Number  | Table number   |
| isEncryptable    | Boolean | True if the table is declared encryptable in the structure file                    |
| isEncrypted      | Boolean | True if the records of the table are encrypted in the data file                    |

(\*) The encryption key can be provided:

- with the `.provideDataKey()` command,
- at the root of a connected device before opening the datastore,
- with the `Discover data key` command.

### Example

You want to know the number of encrypted tables in the current data file:

```

var $status : Object

$status:=ds.encryptionStatus()

If($status.isEncrypted) //the database is encrypted
    C_LONGINT($vcount)
    C_TEXT($tabName)
    For each ($tabName;$status.tables)
        If($status.tables[$tabName].isEncrypted)
            $vcount:=$vcount+1
        End if
    End for each
    ALERT(String($vcount)+" encrypted table(s) in this datastore.")
Else
    ALERT("This database is not encrypted.")
End if

```

## .flushAndLock()

- History

### .flushAndLock()

| Parameter | Type | Description                     |
|-----------|------|---------------------------------|
|           |      | Does not require any parameters |

### Description

The `.flushAndLock()` function flushes the cache of the local datastore and prevents other processes from performing write operations on the database. The datastore is set to a consistent, frozen state. Calling this function is necessary before executing an application snapshot, for example.



This function can only be called:

- on the local datastore (`ds`).
- in client/server environment, on the server machine.

Once this function is executed, write operations such as `.save()` or other `.flushAndLock()` calls are frozen in all other processes until the datastore is unlocked.

When multiple calls to `.flushAndLock()` have been done in the same process, the same number of `.unlock()` calls must be executed to actually unlock the datastore.

The datastore is unlocked when:

- the `.unlock()` function is called in the same process, or
- the process that called the `.flushAndLock()` function is killed.

If the datastore is already locked from another process, the `.flushAndLock()` call is frozen and will be executed when the datastore will be unlocked.

An error is triggered if the `.flushAndLock()` function cannot be executed (e.g. it is run on a remote 4D), .

## CAUTION

Other 4D features and services including [backup](#), [vss](#), and [MSC](#) can also lock the datastore. Before calling `.flushAndLock()`, make sure no other locking action is being used, in order to avoid any unexpected interaction.

### Example

You want to create a copy of the data folder along with its current journal file:

```
$destination:=Folder(fk documents folder).folder("Archive")
$destination.create()

ds.flushAndLock() //Block write operations from other processes

$dataFolder:=Folder(fk data folder)
$dataFolder.copyTo($destination) //Copy the data folder

$oldJournalPath:=New log file //Close the journal and create a new one
$oldJournal:=File($oldJournalPath; fk platform path)
$oldJournal.moveTo($destination) //Save the old journal with data

ds.unlock() //Our copy is over, we can now unlock the datastore
```

### See also

[.locked\(\)](#)  
[.unlock\(\)](#)

## **.getAllRemoteContexts()**

- History

**.getAllRemoteContexts()** : Collection

| Parameter | Type  | Description |
|-----------|---|-------------|
| Result    | Collection <-Collection of optimization context objects |             |

**Advanced mode:** This function is intended for developers who need to customize ORDA default features for specific configurations. In most cases, you will not need to use it.

### Description

The `.getAllRemoteContexts()` function returns a collection of objects containing information on all the active optimization contexts in the datastore.

For more information on how contexts can be created, see [client/server optimization](#).

Each object in the returned collection has the properties listed in the [.getRemoteContextInfo\(\)](#) section.

### Example

The following code sets up two contexts and retrieves them using `.getAllRemoteContexts()`:

```

var $ds : 4D.DataStoreImplementation
var $persons : cs.PersonsSelection
var $addresses : cs.AddressSelection
var $p : cs.PersonsEntity
var $a : cs.AddressEntity
var $contextA; $contextB : Object
var $info : Collection
var $text : Text

// Open remote datastore
$ds:=Open datastore(New object("hostname"; "www.myserver.com"); "myDS")

// Set context A
$contextA:=New object("context"; "contextA")
$persons:=$ds.Persons.all($contextA)
$text:=""
For each ($p; $persons)
    $text:=$p.firstname+" lives in "+$p.address.city+" / "
End for each

// Set context B
$contextB:=New object("context"; "contextB")
$addresses:=$ds.Address.all($contextB)
$text:=""
For each ($a; $addresses)
    $text:=$a.zipCode
End for each

// Get all remote contexts (in this case, contextA and contextB)
$info:=$ds.getAllRemoteContexts()
//$info = [{name:"contextB"; dataclass:"Address"; main:"zipCode"}, {name:"contextA";dataclass:"Persons";main:"firstname,address.city"}]

```

This example serves as a demonstration, it is not meant for real implementation.

## See also

[.getRemoteContextInfo\(\)](#)  
[.setRemoteContextInfo\(\)](#)  
[.clearAllRemoteContexts\(\)](#)

## .getInfo()

- History

**.getInfo(): Object**

| Parameter | Type                          | Description |
|-----------|-------------------------------|-------------|
| Result    | Object<- Datastore properties |             |

### Description

The `.getInfo()` function returns an object providing information about the datastore. This function is useful for setting up generic code.

### Returned object

| <b>Property</b>   | <b>Type</b> | <b>Description</b>   |
|-------------------|-------------|--|
| type              | string      | <ul style="list-style-type: none"> <li>• "4D": main datastore, available through ds</li> <li>• "4D Server": remote datastore, open with Open datastore</li> <li>• True: the datastore is reached through a network connection.</li> </ul>  |
| networked         | boolean     | <ul style="list-style-type: none"> <li>• False: the datastore is not reached through a network connection (local database)</li> </ul>  |
| localID           | text        | <p>ID of the datastore on the machine. Corresponds to the localId string given with the <code>Open datastore</code> command. Empty string ("") for main datastore.</p> <p>Object describing the remote datastore connection (not returned for main datastore). Available properties:</p> |
| <b>Property</b>   | <b>Type</b> | <b>Description</b>   |
| connection object | hostname    | text IP address or name of the remote datastore + ":" + port number  |
|                   | tls         | boolean True if secured connection is used with the remote datastore   |
|                   | idleTimeout | number Session inactivity timeout (in minutes)   |
|                   | user        | text User authenticated on the remote datastore  |

- If the `.getInfo()` function is executed on a 4D Server or 4D single-user, `networked` is False.
- If the `.getInfo()` function is executed on a remote 4D, `networked` is True

## Example 1

```
var $info : Object

$info:=ds.getInfo() //Executed on 4D Server or 4D
//{"type":"4D","networked":false,"localID":""}

$info:=ds.getInfo() // Executed on 4D remote
//{"type":"4D","networked":true,"localID":""}
```

## Example 2

On a remote datastore:

```
var $remoteDS : cs.DataStore
var $info; $connectTo : Object

$connectTo:=New
object("hostname";"111.222.33.44:8044";"user";"marie";"password";"aaaa"
$remoteDS:=Open datastore($connectTo;"students")
$info:=$remoteDS.getInfo()

// {"type":"4D Server",
// "localID":"students",
// "networked":true,
// "connection":
//{
// hostname:"111.222.33.44:8044", "tls":false, "idleTimeout":2880, "user":"m
```

## .getRemoteContextInfo()

- History

**.getRemoteContextInfo(contextName : Text) : Object**

| Parameter       | Type                 | Description                             |
|-----------------|----------------------|---|
| contextNameText | -> Text              | Name of the context                     |
| Result          | Object <- Collection | Description of the optimization context |

**Advanced mode:** This function is intended for developers who need to customize ORDA default features for specific configurations. In most cases, you will not need to use it.

## Description

The `.getRemoteContextInfo()` function returns an object that holds information on the *contextName* optimization context in the datastore..

For more information on how optimization contexts can be created, see [client/server optimization](#).

## Returned object

The returned object has the following properties:

| Property                  | Type | Description   |
|---------------------------|------|---|
| name                      | Text | Name of the context   |
| main                      | Text | Attribute(s) associated to the context (attribute names are separated by a comma)   |
| dataclass                 | Text | Dataclass name  |
| currentItem<br>(optional) | Text | The attributes of the <a href="#">page mode</a> if the context is linked to a list box. Returned as <code>Null</code> or empty text element if the context name is not used for a list box, or if there is no context for the currentItem |

Since contexts behave as filters for attributes, if *main* is returned empty, it means that no filter is applied, and that the server returns all the dataclass attributes.

## Example

See the example from the [.setRemoteContextInfo\(\)](#) section.

## See also

[.setRemoteContextInfo\(\)](#)  
[.getAllRemoteContexts\(\)](#)  
[.clearAllRemoteContexts\(\)](#)

## .getRequestLog()

- History

**.getRequestLog() :** Collection

| Parameter | Type                     | Description  |
|-----------|--------------------------|--|
| Result    | Collection <- Collection | Collection of objects, where each object describes a request |

## Description

The `.getRequestLog()` function returns the ORDA requests logged in memory on the client side. The ORDA request logging must have previously been enabled using the

[`.startRequestLog\(\)`](#) function.

This function must be called on a remote 4D, otherwise it returns an empty collection. It is designed for debugging purposes in client/server configurations.

### Returned value

Collection of stacked request objects. The most recent request has index 0.

For a description of the ORDA request log format, please refer to the [ORDA client requests](#) section.

### Example

See Example 2 of [`.startRequestLog\(\)`](#).

## **.isAdminProtected()**

- History

**.isAdminProtected()** : Boolean

| Parameter | Type       | Description   |
|-----------|------------|---|
| Result    | Boolean <- | True if the Data Explorer access is disabled, False if it is enabled<br>(default) |

### Description

The `.isAdminProtected()` function returns `True` if [Data Explorer](#) access has been disabled for the working session.

By default, the Data Explorer access is granted for `webAdmin` sessions, but it can be disabled to prevent any data access from administrators (see the [`.setAdminProtection\(\)`](#) function).

### See also

[`.setAdminProtection\(\)`](#)

## **.locked()**

- History

**.locked()** : Boolean

| Parameter | Type       | Description    |
|-----------|------------|----------------|
| Result    | Boolean <- | True if locked |

### Description

The `.locked()` function returns True if the local datastore is currently locked.

You can lock the datastore using the [`.flushAndLock\(\)`](#) function before executing a snapshot of the data file, for example.



CAUTION

The function will also return `True` if the datastore was locked by another administration feature such as backup or vss (see [.flushAndLock\(\)](#)).

## See also

[.flushAndLock\(\)](#)  
[.unlock\(\)](#)

## .makeSelectionsAlterable()

- History

### .makeSelectionsAlterable()

| Parameter | Type | Description                     |
|-----------|------|---------------------------------|
|           |      | Does not require any parameters |

### Description

The `.makeSelectionsAlterable()` function sets all entity selections as alterable by default in the current application datastores (including [remote datastores](#)). It is intended to be used once, for example in the `On Startup` database method.

When this function is not called, new entity selections can be shareable, depending on the nature of their "parent", or [how they are created](#).

This function does not modify entity selections created by [.copy\(\)](#) or [OB Copy](#) when the explicit `ck shared` option is used.

**Compatibility:** This function must only be used in projects converted from 4D versions prior to 4D v18 R5 and containing [.add\(\)](#) calls. In this context, using `.makeSelectionsAlterable()` can save time by restoring instantaneously the previous 4D behavior in existing projects. On the other hand, using this method in new projects created in 4D v18 R5 and higher **is not recommended**, since it prevents entity selections to be shared, which provides greater performance and scalability.

## .provideDataKey()

- History

**.provideDataKey( curPassPhrase : Text ) : Object**  
**.provideDataKey( curDataKey : Object ) : Object**

| Parameter     | Type   | Description                              |
|---------------|--------|--|
| curPassPhrase | Text   | -> Current encryption passphrase         |
| curDataKey    | Object | -> Current data encryption key           |
| Result        | Object | <- Result of the encryption key matching |

### Description

The `.provideDataKey()` function allows providing a data encryption key for the current data file of the datastore and detects if the key matches the encrypted data. This function can be used when opening an encrypted database, or when executing any encryption operation that requires the encryption key, such as re-encrypting the

data file.

- The `.provideDataKey()` function must be called in an encrypted database. If it is called in a non-encrypted database, the error 2003 (the encryption key does not match the data.) is returned. Use the `Data file encryption status` command to determine if the database is encrypted.
- The `.provideDataKey()` function cannot be called from a remote 4D or an encrypted remote datastore.

If you use the `curPassPhrase` parameter, pass the string used to generate the data encryption key. When you use this parameter, an encryption key is generated.

If you use the `curDataKey` parameter, pass an object (with `encodedKey` property) that contains the data encryption key. This key may have been generated with the `New data key` command.

If a valid data encryption key is provided, it is added to the `keyChain` in memory and the encryption mode is enabled:

- all data modifications in encryptable tables are encrypted on disk (.4DD, .journal, 4Dindx files)
- all data loaded from encryptable tables is decrypted in memory

## Result

The result of the command is described in the returned object:

| Property                  | Type       | Description   |
|---------------------------|------------|---|
| success                   | Boolean    | True if the provided encryption key matches the encrypted data, False otherwise |
|                           |            | Properties below are returned only if success is FALSE                          |
| status                    | Number     | Error code (4 if the provided encryption key is wrong)                          |
| statusText                | Text       | Error message   |
| errors                    | Collection | Stack of errors. The first error has the highest index                          |
| [<br>].componentSignature | Text       | Internal component name   |
| [ ].errCode               | Number     | Error number  |
| [ ].message               | Text       | Error message   |

If no `curPassphrase` or `curDataKey` is given, `.provideDataKey()` returns `null` (no error is generated).

## Example

```

var $keyStatus : Object
var $passphrase : Text

$passphrase:=Request("Enter the passphrase")
If (OK=1)
    $keyStatus:=ds.provideDataKey($passphrase)
    If ($keyStatus.success)
        ALERT("You have provided a valid encryption key")
    Else
        ALERT("You have provided an invalid encryption key, you will
not be able to work with encrypted data")
    End if
End if

```

## .setAdminProtection()

- History

**.setAdminProtection( *status* : Boolean )**

| Parameter | Type    | Description   |
|-----------|---------|---|
| status    | Boolean | - True to disable Data Explorer access to data on the <code>webAdmin</code> port, False (default) to grant access |

### Description

The `.setAdminProtection()` function allows disabling any data access on the [web admin port](#), including for the [Data Explorer](#) in [WebAdmin](#) sessions.

By default when the function is not called, access to data is always granted on the web administration port for a session with [WebAdmin](#) privilege using the Data Explorer. In some configurations, for example when the application server is hosted on a third-party machine, you might not want the administrator to be able to view your data, although they can edit the server configuration, including the [access key](#) settings.

In this case, you can call this function to disable the data access from Data Explorer on the web admin port of the machine, even if the user session has the [WebAdmin](#) privilege. When this function is executed, the data file is immediately protected and the status is stored on disk: the data file will be protected even if the application is restarted.

### Example

You create a `protectDataFile` project method to call before deployments for example:

```
ds.setAdminProtection(True) //Disables the Data Explorer data access
```

### See also

[.isAdminProtected\(\)](#)

## .setRemoteContextInfo()

- History

**.setRemoteContextInfo( *contextName* : Text ; *dataClassName* : Text ; *attributes* : Text {; *contextType* : Text { ; *pageLength* : Integer}})**  
**.setRemoteContextInfo( *contextName* : Text ; *dataClassName* : Text; *attributesColl* : Collection {; *contextType* : Text { ; *pageLength* : Integer }} )**

```

.setRemoteContextInfo( contextName : Text ; dataClassObject : 4D.DataClass ;
attributes : Text {; contextType : Text { ; pageLength : Integer }})
.setRemoteContextInfo( contextName : Text ; dataClassObject : 4D.DataClass ;
attributesColl : Collection {; contextType : Text { ; pageLength : Integer }} )

```

| Parameter       | Type         | Description   |
|-----------------|--------------|---|
| contextName     | Text         | <ul style="list-style-type: none"> <li>- Name of the context</li> <li>&gt;</li> </ul>   |
| dataClassName   | Text         | <ul style="list-style-type: none"> <li>- Name of the dataclass</li> <li>&gt;</li> </ul>   |
| dataClassObject | 4D.DataClass | <ul style="list-style-type: none"> <li>- dataclass object (e.g datastore.Employee)</li> <li>&gt;</li> </ul>                                 |
| attributes      | Text         | <ul style="list-style-type: none"> <li>- Attribute list separated by a comma</li> <li>&gt;</li> </ul>                                       |
| attributesColl  | Collection   | <ul style="list-style-type: none"> <li>- Collection of attribute names (text)</li> <li>&gt;</li> </ul>                                      |
| contextType     | Text         | <ul style="list-style-type: none"> <li>- If provided, value must be "main" or "currentItem"</li> <li>&gt;</li> </ul>                        |
| pageLength      | Integer      | <ul style="list-style-type: none"> <li>- Page length of the entity selection linked to the context (default is 80)</li> <li>&gt;</li> </ul> |

**Advanced mode:** This function is intended for developers who need to customize ORDA default features for specific configurations. In most cases, you will not need to use it.

## Description

The `.setRemoteContextInfo()` function links the specified dataclass attributes to the *contextName* optimization context. If an optimization context already exists for the specified attributes, this command replaces it.

When you pass a context to the ORDA class functions, the REST request optimization is triggered immediately:

- the first entity is not fully loaded as done in automatic mode
- pages of 80 entities (or `pageLength` entities) are immediately asked to the server with only the attributes in the context

For more information on how optimization contexts are built, refer to the [client/server optimization paragraph](#)

In *contextName*, pass the name of the optimization context to link to the dataclass attributes.

To designate the dataclass that will receive the context, you can pass a *dataClassName* or a *dataClassObject*.

To designate the attributes to link to the context, pass either a list of attributes separated by a comma in *attributes* (Text), or a collection of attribute names in *attributesColl* (collection of text).

If *attributes* is an empty Text, or *attributesColl* is an empty collection, all the scalar attributes of the dataclass are put in the optimization context. If you pass an attribute that does not exist in the dataclass, the function ignores it and an error is thrown.

You can pass a *contextType* to specify if the context is a standard context or the context of the current entity selection item displayed in a list box:

- If set to "main" (default), the *contextName* designates a standard context.
- If set to "currentItem", the attributes passed are put in the context of the current item. See [Entity selection-based list box](#).

In *pageLength*, specify the number of dataclass entities to request from the server.

You can pass a *pageLength* for a relation attribute which is an entity selection (one to many). The syntax is `relationAttributeName:pageLength` (e.g `employees:20`).

### Example 1

```
var $ds : 4D.DataStoreImplementation
var $persons : cs.PersonsSelection
var $p : cs.PersonsEntity
var $contextA : Object
var $info : Object
var $text : Text

// Open remote datastore
$ds:=Open datastore(New object("hostname"; "www.myserver.com"); "myDS")

// Set context info
$contextA:=New object("context"; "contextA")
$ds.setRemoteContextInfo("contextA"; $ds.Persons; "firstname,
lastname")

// Send requests to the server using a loop
$persons:=$ds.Persons.all($contextA)
$text:=""
For each ($p; $persons)
  $text:=$p.firstname + " " + $p.lastname
End for each

// Check contents of the context
$info:=$ds.getRemoteContextInfo("contextA")
// $info = {name:"contextA";dataclass:"Persons";main:"firstname,
lastname"}
```

This example serves as a demonstration, it is not meant for real implementation.

### Example 2

The following piece of code requests pages of 30 entities of the `Address` dataclass from the server. The returned entities only contain the `zipCode` attribute.

For each `Address` entity, 20 Persons entities are returned, and they only contain the `lastname` and `firstname` attributes:

```
var $ds : 4D.DataStoreImplementation

$ds:=Open datastore(New object("hostname"; "www.myserver.com"); "myDS")

$ds.setRemoteContextInfo("contextA"; $ds.Address; "zipCode,
persons:20,\npersons.lastname, persons.firstname"; "main"; 30)
```

### Example 3 - Listbox

```

// When the form loads
Case of
    : (Form event code=On Load)

        Form.ds:=Open datastore(New object("hostname"; "www.myserver.co
"myDS"))

        // Set the attributes of the page context
        Form.ds.setRemoteContextInfo("LB"; Form.ds.Persons; "age, gende
children"; "currentItem")

        Form.settings:=New object("context"; "LB")
        Form.persons:=Form.ds.Persons.all(Form.settings)
        // Form.persons is displayed in a list box
End case

// When you get the attributes in the context of the current item:
Form.currentItemLearntAttributes:=Form.selectedPerson.getRemoteContextA
// Form.currentItemLearntAttributes = "age, gender, children"

```

## See also

[.getRemoteContextInfo\(\)](#)  
[.getAllRemoteContexts\(\)](#)  
[.clearAllRemoteContexts\(\)](#)

## .startRequestLog()

- History

**.startRequestLog()**  
**.startRequestLog( file : 4D.File )**  
**.startRequestLog( file : 4D.File ; options : Integer )**  
**.startRequestLog( reqNum : Integer )**

| Parameter | Type   | Description |
|-----------|--|-------------|
| file      | 4D.File -> File object                                       |             |
| options   | Integer-> Log response option (server only)                  |             |
| reqNum    | Integer-> Number of requests to keep in memory (client only) |             |

## Description

The `.startRequestLog()` function starts the logging of ORDA requests on the client side or on the server side. It is designed for debugging purposes in client/server configurations.

### (!) INFO

For a description of the ORDA request log format, please refer to the [ORDA requests](#) section.

## Client-side

To create a client-side ORDA request log, call this function on a remote machine. The log can be sent to a file or to memory, depending on the parameter type:

- If you passed a `file` object created with the `File` command, the log data is

written in this file as a collection of objects (JSON format). Each object represents a request.

If the file does not already exist, it is created. Otherwise if the file already exists, the new log data is appended to it. If `.startRequestLog()` is called with a file while a logging was previously started in memory, the memory log is stopped and emptied.

A `]` character must be manually appended at the end of the file to perform a JSON validation

- If you passed a *reqNum* integer, the log in memory is emptied (if any) and a new log is initialized. It will keep *reqNum* requests in memory until the number is reached, in which case the oldest entries are emptied (FIFO stack).  
If `.startRequestLog()` is called with a *reqNum* while a logging was previously started in a file, the file logging is stopped.
- If you did not pass any parameter, the log is started in memory. If `.startRequestLog()` was previously called with a *reqNum* (before a `.stopRequestLog()`), the log data is stacked in memory until the next time the log is emptied or `.stopRequestLog()` is called.

## Server-side

To create a server-side ORDA request log, call this function on the server machine. The log data is written in a file in `.jsonl` format. Each object represents a request. If the file does not already exist, it is created. Otherwise if the file already exists, the new log data is appended to it.

- If you passed the *file* parameter, the log data is written in this file, at the requested location. - If you omit the *file* parameter or if it is null, the log data is written in a file named *ordaRequests.jsonl* and stored in the "/LOGS" folder.
- The *options* parameter can be used to specify if the server response has to be logged, and if it should include the body. By default when the parameter is omitted, the full response is logged. The following constants can be used in this parameter:

| Constant                      | Description                               |
|-------------------------------|---|
| srl log all                   | Log the response entirely (default value) |
| srl log no response           | Disable the logging of the response       |
| srl log response without body | Log the response without the body         |

## Example 1

You want to log ORDA client requests in a file and use the log sequence number:

```
var $file : 4D.File
var $e : cs.PersonsEntity

$file:=File("/LOGS/ORDAResponses.txt") //logs folder

SET DATABASE PARAMETER(Client Log Recording;1) //to trigger the global
log sequence number
ds.startRequestLog($file)
$e:=ds.Persons.get(30001) //send a request
ds.stopRequestLog()
SET DATABASE PARAMETER(Client Log Recording;0)
```

## Example 2

You want to log ORDA client requests in memory:

```
var $es : cs.PersonsSelection
var $log : Collection

ds.startRequestLog(3) //keep 3 requests in memory

$es:=ds.Persons.query("name=:1";"Marie")
$es:=ds.Persons.query("name IN :1";New collection("Marie"))
$es:=ds.Persons.query("name=:1";"So@")

$log:=ds.getRequestLog()
ALERT("The longest request lasted: "+String($log.max("duration"))+" ms")
```

## Example 3

You want to log ORDA server requests in a specific file and enable the log sequence number and duration:

```
SET DATABASE PARAMETER(4D Server Log Recording;1)

$file:=Folder(fk logs folder).file("myOrdaLog.jsonl")
ds.startRequestLog($file)
...
ds.stopRequestLog()
SET DATABASE PARAMETER(4D Server Log Recording;0)
```

## .startTransaction()

- History

### .startTransaction()

| Parameter Type | Description                     |
|----------------|---------------------------------|
|                | Does not require any parameters |

### Description

The `.startTransaction()` function starts a transaction in the current process on the database matching the datastore to which it applies. Any changes made to the datastore's entities in the transaction's process are temporarily stored until the transaction is either validated or cancelled.

If this method is called on the main datastore (i.e. the datastore returned by the `ds` command), the transaction is applied to all operations performed on the main datastore and on the underlying database, thus including ORDA and classic languages.

You can nest several transactions (sub-transactions). Each transaction or sub-transaction must eventually be cancelled or validated. Note that if the main transaction is cancelled, all of its sub-transactions are also cancelled even if they were validated individually using the `.validateTransaction()` function.

## Example

```

var $connect; $status : Object
var $person : cs.PersonsEntity
var $ds : cs.DataStore
var $choice : Text
var $error : Boolean

Case of
    : ($choice="local")
        $ds:=ds
    : ($choice="remote")
        $connect:=New object("hostname";"111.222.3.4:8044")
        $ds:=Open datastore($connect;"myRemoteDS")
End case

$ds.startTransaction()
$person:=$ds.Persons.query("lastname=:1";"Peters").first()

If ($person#Null)
    $person.lastname:="Smith"
    $status:=$person.save()
End if
...
...
If ($error)
    $ds.cancelTransaction()
Else
    $ds.validateTransaction()
End if

```

## **.stopRequestLog()**

- History

### **.stopRequestLog()**

| <b>Parameter</b> | <b>Type</b> | <b>Description</b>              |
|------------------|-------------|---------------------------------|
|                  |             | Does not require any parameters |

### **Description**

The `.stopRequestLog()` function stops any logging of ORDA requests on the machine it is called (client or server).

It actually closes the opened document on disk. On the client side, if the log was started in memory, it is stopped.

This function does nothing if logging of ORDA requests was not started on the machine.

### **Example**

See examples for [`.startRequestLog\(\)`](#).

## **.unlock()**

- History

### **.unlock()**

| Parameter Type | Description                     |
|----------------|---------------------------------|
|                | Does not require any parameters |

## Description

The `.unlock()` function removes the current lock on write operations in the datastore, if it has been set in the same process. Write operations can be locked in the local datastore using the [`.flushAndLock\(\)`](#) function.

If the current lock was the only lock on the datastore, write operations are immediately enabled. If the `.flushAndLock()` function was called several times in the process, the same number of `.unlock()` must be called to actually unlock the datastore.

The `.unlock()` function must be called from the process that called the corresponding `.flushAndLock()`, otherwise the function does nothing and the lock is not removed.

If the `.unlock()` function is called in an unlocked datastore, it does nothing.

## See also

[`.flushAndLock\(\)`](#)  
[`.locked\(\)`](#)

## **.validateTransaction()**

- History

### **.validateTransaction()**

| Parameter Type | Description                     |
|----------------|---------------------------------|
|                | Does not require any parameters |

## Description

The `.validateTransaction()` function accepts the transaction that was started with [`.startTransaction\(\)`](#) at the corresponding level on the specified datastore.

The function saves the changes to the data on the datastore that occurred during the transaction.

You can nest several transactions (sub-transactions). If the main transaction is cancelled, all of its sub-transactions are also cancelled, even if they were validated individually using this function.

## Example

See example for [`.startTransaction\(\)`](#).

## Email

Creating, sending or receiving emails in 4D is done by handling an `Email` object.

`Email` objects are created when receiving mails through a *transporter* class function:

- IMAP - `.getMail()` and `.getMails()` functions to get emails from an IMAP server
- POP3 - `.getMail()` function to get an email from a POP3 server.

You can also create a new, blank `Email` object by calling the [New object](#) 4D command, and then fill it with [Email object properties](#).

You send `Email` objects using the SMTP `.send()` function.

[MAIL Convert from MIME](#) and [MAIL Convert to MIME](#) commands can be used to convert `Email` objects to and from MIME contents.

### Email Object

Email objects provide the following properties:

4D follows the [JMAP specification](#) to format the Email object.

**.attachments : Collection** collection of `4D.MailAttachment` object(s)

**.bcc : Text**

**.bcc : Object**

**.bcc : Collection** Blind Carbon Copy (BCC) hidden email recipient [addresse\(s\)](#) of the email

**.bodyStructure : Object** *EmailBodyPart* object, i.e. the full MIME structure of the message body (optional)

**.bodyValues : Object** *EmailBodyValue* object, containing an object for each ¥  
<partID> of `bodyStructure` (optional)

**.cc : Text**

**.cc : Object**

**.cc : Collection** Carbon Copy (CC) additional email recipient [addresse\(s\)](#) of the email

**.comments : Text** additional comments header

**.from : Text**

**.from : Object**

**.from : Collection** Originating [address\(es\)](#) of the email

**.headers : Collection** collection of `EmailHeader` objects, in the order they appear in the message

**.htmlBody : Text** HTML representation of the email message (default charset is UTF-8) (optional, SMTP only)

**.id : Text** unique ID from the IMAP server

**.inReplyTo : Text** message identifier(s) of the original message(s) to which the current message is a reply

**.keywords : Object** set of keywords as an object, where each property name is a keyword and each value is true

**.messageId : Text** message identifier header ("message-id")

**.receivedAt : Text** timestamp of the email's arrival on the IMAP server in ISO 8601 UTC format (ex: 2020-09-13T16:11:53Z)

**.references : Collection** Collection of all message-ids of messages in the preceding reply chain

**.replyTo : Text**

**.replyTo : Object**

**.replyTo : Collection** [addresse\(s\)](#) for responses

**.sendAt : Text** Email timestamp in ISO 8601 UTC format

**.sender : Text**

**.sender : Object**

**.sender : Collection** email source [addresse\(s\)](#) of the email

**.size : Integer** size (expressed in bytes) of the Email object returned by the IMAP server

**.subject : Text** description of topic

**.textBody : Text** Plain text representation of the email message (default charset is UTF-8) (optional, SMTP only)

**.to : Text**

**.to : Object**

**.to : Collection** primary recipient [addresse\(s\)](#) of the email

## Email Addresses

All properties that contain email addresses (`from`, `cc`, `bcc`, `to`, `sender`, `replyTo`) accept a value of text, object, or collection type.

### Text

- single email: "[somebody@domain.com](mailto:somebody@domain.com)"

- single display name+email: "Somebody [somebody@domain.com](mailto:somebody@domain.com)"
- several emails: "Somebody [somebody@domain.com](mailto:somebody@domain.com),[me@home.org](mailto:me@home.org)"

## Object

An object with two properties:

| Property | Type | Description                |
|----------|------|----------------------------|
| name     | Text | Display name (can be null) |
| email    | Text | Email address              |

## Collection

A collection of address objects.

## Handling body part

The `textBody` and `htmlBody` properties are only used with the `SMTP.send()` function to allow sending simple mails. When both property are filled, the MIME content-type multipart/alternative is used. The email client should then recognize the multipart/alternative part and display the text part or html part as necessary.

`bodyStructure` and `bodyValues` are used for `SMTP` when the `Email object` is built from a MIME document, e.g. when generated by the `MAIL Convert from MIME` command. In this case, both `bodyStructure` and `bodyValues` properties must be passed together, and it is not recommended to use `textBody` and `htmlBody`.

## Example of bodyStructure and bodyValues objects

```
"bodyStructure": {
  "type": "multipart/mixed",
  "subParts": [
    {
      "partId": "p0001",
      "type": "text/plain"
    },
    {
      "partId": "p0002",
      "type": "text/html"
    }
  ]
},
"bodyValues": {
  "p0001": {
    "value": "I have the most brilliant plan. Let me tell you all about it."
  },
  "p0002": {
    "value": "<!DOCTYPE html><html><head><title></title><style type='text/css'>div{font-size:16px}</style></head><body><div>I have the most brilliant plan. Let me tell you all about it.</div></body></html>"
  }
}
```

## .attachments

**.attachments** : Collection

## Description

The `.attachments` property contains a collection of `4D.MailAttachment` object(s).

Attachment objects are defined through the [MAIL New attachment](#) command.

Attachment objects have specific [properties and functions](#).

## .bcc

**.bcc** : Text

**.bcc** : Object

**.bcc** : Collection

## Description

The `.bcc` property contains the Blind Carbon Copy (BCC) hidden email recipient [adresse\(s\)](#) of the email.

## .bodyStructure

**.bodyStructure** : Object

## Description

The `.bodyStructure` property contains the *EmailBodyPart* object, i.e. the full MIME structure of the message body (optional). See [Handling body part](#) section.

The `.bodyStructure` object contains the following properties:

| Property    | Type                  | Value   |
|-------------|-----------------------|---|
| partID      | Text                  | Identifies the part uniquely within the email   |
| type        | Text                  | (mandatory) Value of the Content-Type header field of the part  |
| charset     | Text                  | Value of the charset parameter of the Content-Type header field   |
| encoding    | Text                  | If <code>isEncodingProblem=true</code> , the Content-Transfer-Encoding value is added (by default undefined)  |
| disposition | Text                  | Value of the Content-Disposition header field of the part   |
| language    | Collection of texts   | List of language tags, as defined in <a href="#">RFC3282</a> , in the Content-Language header field of the part, if present.                                    |
| location    | Text                  | URI, as defined in <a href="#">RFC2557</a> , in the Content-Location header field of the part, if present.  |
| subParts    | Collection of objects | Body parts of each child (collection of <i>EmailBodyPart</i> objects)   |
| headers     | Collection of objects | List of all header fields in the part, in the order they appear in the message (collection of <i>EmailHeader</i> objects, see <a href="#">headers</a> property) |

## .bodyValues

**.bodyValues** : Object

## Description

The `.bodyValues` property contains the *EmailBodyValue* object, containing an object for each `¥<partID>` of `bodyStructure` (optional). See [Handling body part](#) section.

The `.bodyValues` object contains the following properties:

| Property                              | Type    | Value   |
|---------------------------------------|---------|---|
| <code>partID.value</code>             | text    | Value of the body part<br>True if malformed sections are found while decoding           |
| <code>partID.isEncodingProblem</code> | boolean | the charset, or unknown charset, or unknown content transfer-encoding. False by default |

## .CC

`.cc` : Text  
`.cc` : Object  
`.cc` : Collection

### Description

The `.cc` property contains the Carbon Copy (CC) additional email recipient [adresse\(s\)](#) of the email.

## .comments

`.comments` : Text

### Description

The `.comments` property contains an additional comments header.

Comments only appear within the header section of the message (keeping the message's body untouched).

For specific formatting requirements, please consult the [RFC#5322](#).

## .from

`.from` : Text  
`.from` : Object  
`.from` : Collection

### Description

The `.from` property contains the Originating [address\(es\)](#) of the email.

Each email you send out has both the [sender](#) and **from** addresses:

- the sender domain is what the receiving email server gets when opening the session,
- the from address is what the recipient(s) will see.

For better deliverability, it is recommended to use the same from and sender addresses.

## .headers

`.headers` : Collection

### Description

The `.headers` property contains a collection of `EmailHeader` objects, in the order they appear in the message. This property allows users to add extended (registered)

headers or user-defined (not registered, starting with "X") headers.

If an `EmailHeader` object property defines a header such as "from" or "cc" which is already set as a property at the mail level, the `EmailHeader` property is ignored.

Every object of the headers collection can contain the following properties:

| <b>Property</b>        | <b>Type</b> | <b>Value</b>   |
|------------------------|-------------|--|
| <code>[]</code> .name  | text        | (mandatory) Header field name as defined in <a href="#">RFC#5322</a> . If null or undefined, the header field is not added to the MIME header. |
| <code>[]</code> .value | text        | Header field values as defined in <a href="#">RFC#5322</a>   |

## **.htmlBody**

**.htmlBody** : Text

### **Description**

The `.htmlBody` property contains the HTML representation of the email message (default charset is UTF-8) (optional, SMTP only). See [Handling body part](#) section.

## **.id**

**.id** : Text

### **Description**

[IMAP transporter](#) only.

The `.id` property contains the unique ID from the IMAP server.

## **.inReplyTo**

**.inReplyTo** : Text

### **Description**

The `.inReplyTo` property contains the message identifier(s) of the original message(s) to which the current message is a reply.

For specific formatting requirements, please consult the [RFC#5322](#).

## **.keywords**

**.keywords** : Object

### **Description**

The `.keywords` property contains a set of keywords as an object, where each property name is a keyword and each value is true.

This property is the "keywords" header (see [RFC#4021](#)).

| <b>Property</b>                   | <b>Type</b> | <b>Value</b>                        |
|-----------------------------------|-------------|-------------------------------------|
| <code>.\${&lt;keyword&gt;}</code> | boolean     | Keyword to set (value must be true) |

Reserved keywords:

- \$draft - Indicates a message is a draft
- \$seen - Indicates a message has been read
- \$flagged - Indicates a message needs special attention (e.g., Urgent)
- \$answered - Indicates a message has been replied to
- \$deleted - Indicates a message to delete

## Example

```
$mail.keywords["$flagged"] :=True  
$mail.keywords["4d"] :=True
```

## .messageId

**.messageId** : Text

### Description

The `.messageId` property contains a message identifier header ("message-id").

This header is usually "lettersOrNumbers@domainname", e.g. "[abcdef.123456@4d.com](mailto:abcdef.123456@4d.com)". This unique ID is used in particular on forums or public mailing lists. In general, mail servers automatically add this header to the messages they send.

## .receivedAt

**.receivedAt** : Text

### Description

[IMAP transporter](#) only.

The `.receivedAt` property contains the timestamp of the email's arrival on the IMAP server in ISO 8601 UTC format (ex: 2020-09-13T16:11:53Z).

## .references

**.references** : Collection

### Description

The `.references` property contains the Collection of all message-ids of messages in the preceding reply chain.

For specific formatting requirements, please consult the [RFC#5322](#).

## .replyTo

**.replyTo** : Text  
**.replyTo** : Object  
**.replyTo** : Collection

### Description

The `.replyTo` property contains the [addresse\(s\)](#) for responses.

## .sendAt

.sendAt : Text

### Description

The `.sendAt` property contains the Email timestamp in ISO 8601 UTC format.

## .sender

.sender : Text

.sender : Object

.sender : Collection

### Description

The `.sender` property contains the email source [addresse\(s\)](#) of the email.

Each email you send out has both the **sender** and [from](#) addresses:

- the sender domain is what the receiving email server gets when opening the session,
- the from address is what the recipient(s) will see.

For better deliverability, it is recommended to use the same from and sender addresses.

## .size

.size : Integer

### Description

[IMAP transporter](#) only.

The `.size` property contains the size (expressed in bytes) of the Email object returned by the IMAP server.

## .subject

.subject : Text

### Description

The `.subject` property contains the description of topic.

## .textBody

.textBody : Text

### Description

The `.textBody` property contains the Plain text representation of the email message (default charset is UTF-8) (optional, SMTP only). See [Handling body part](#) section.

## .to

.to : Text  
.to : Object  
.to : Collection

## Description

The `.to` property contains the primary recipient [addresse\(s\)](#) of the email.

## MAIL Convert from MIME

- History

**MAIL Convert from MIME**( *mime* : Blob ) : Object  
**MAIL Convert from MIME**( *mime* : Text ) : Object

| Parameter   | Type                      | Description    |
|-------------|---------------------------|----------------|
| <i>mime</i> | Blob, Text->Email in MIME |                |
| Result      | Object                    | <-Email object |

## Description

The `MAIL Convert from MIME` command converts a MIME document into a valid email object.

4D follows the [JMAP specification](#) to format the returned email object.

Pass in *mime* a valid MIME document to convert. It can be provided by any mail server or application. You can pass a BLOB or a text *mime* parameter. If the MIME comes from a file, it is recommended to use a BLOB parameter to avoid issues related to charset and line break conversions.

## Returned object

Email object.

## Example 1

You want to load a mail template saved as MIME in a text document and send an email:

```
var $mime: Blob
var $mail;$server;$transporter;$status: Object

$mime:=File("/PACKAGE/Mails/templateMail.txt").getContent()

$mail:=MAIL Convert from MIME($mime)
$mail.to:="smith@mail.com"
$mail.subject:="Hello world"

$server:=New object
$server.host:="smtp.gmail.com"
$server.port:=465
$server.user:="test@gmail.com"
$server.password:="XXXX"

$transporter:=SMTP New transporter($server)
$status:=$transporter.send($mail)
```

## Example 2

In this example, you send directly a 4D Write Pro document containing pictures:

```
var $mime: Blob
var $email;$server;$transporter;$status: Object

// Mime export of the 4D Write Pro document
WP EXPORT VARIABLE(WParea;$mime;wk mime html)

// convert 4D Write Pro Mime variable in mail object
$email:=MAIL Convert from MIME($mime)

// Fill your mail object headers
$email.subject:="4D Write Pro HTML body"
$email.from:="YourEmail@gmail.com"
$email.to:="RecipientEmail@mail.com"

$server:=New object
$server.host:="smtp.gmail.com"
$server.port:=465
$server.user:="YourEmail@gmail.com"
$server.password:="XXXX"

$transporter:=SMTP New transporter($server)
$status:=$transporter.send($email)
```

## MAIL Convert to MIME

- History

**MAIL Convert to MIME(** *mail* : Object { ; *options* : Object } **) : Text**

| Parameter Type | Description                               |
|----------------|---|
| mail           | Object->Email object                      |
| options        | Object->Charset and encoding mail options |
| Result         | Text <-Email object converted to MIME     |

### Description

The `MAIL Convert to MIME` command converts an email object into MIME text. This command is called internally by [SMTP\\_transporter.send\(\)](#) to format the email object before sending it. It can be used to analyze the MIME format of the object.

In *mail*, pass the content and the structure details of the email to convert. This includes information such as the email addresses (sender and recipient(s)), the message itself, and the type of display for the message.

4D follows the [JMAP specification](#) to format the email object.

In *options*, you can set a specific charset and encoding configuration for the mail. The following properties are available:

| Property      | Type | Description  |                      |   |
|---------------|------|--|----------------------|---|
|               |      | Charset and encoding used for the following parts of the email:<br>subject, attachment filenames, and email name attribute(s).<br>Possible values: |                      |   |
|               |      | Constant   | Value                | Comment   |
| headerCharset | Text | mail mode  | US-ASCII ISO-2022-JP | <ul style="list-style-type: none"> <li><i>headerCharset</i>: US-ASCII if possible, Japanese (ISO-2022-JP) &amp; Quoted-printable if possible, otherwise UTF-8 &amp; Quoted-printable</li> <li><i>bodyCharset</i>: US-ASCII if possible, Japanese (ISO-2022-JP) &amp; 7-bit if possible, otherwise UTF-8 &amp; Quoted-printable</li> </ul> |
| bodyCharset   | Text | mail mode  | ISO-8859-1           | <ul style="list-style-type: none"> <li><i>headerCharset</i>: ISO-8859-1 &amp; Quoted-printable</li> <li><i>bodyCharset</i>: ISO-8859-1 &amp; 8-bit</li> </ul>   |
|               |      | mail mode  | US-UTF8              | <i>headerCharset</i> & <i>bodyCharset</i> : US-ASCII if possible, otherwise UTF-8 & Quoted-printable ( <b>default value</b> )   |
|               |      | mail mode  | US-UTF8 in base64    | <i>headerCharset</i> & <i>bodyCharset</i> : US-ASCII if possible, otherwise UTF-8 & base64  |
| bodyCharset   | Text | Charset and encoding used for the html and text body contents of the email. Possible values: Same as for headerCharset (see above)                 |                      |   |

If the *options* parameter is omitted, the mail mode UTF8 configuration is used for header and body parts.

## Example

```
var $mail: Object
var $mime: Text
$mail:=New object

// Creation of a mail
$mail.from:="tsales@massmarket.com"
$mail.subject:="Terrific Sale! This week only!"
$mail.textBody:="Text format email"
$mail.htmlBody:="<html><body>HTML format email</body></html>"
$mail.to:=New collection
$mail.to.push(New object ("email";"noreply@4d.com"))
$mail.to.push(New object ("email";"test@4d.com"))

// transform the mail object in MIME
$mime:=MAIL Convert to MIME($mail)

// Contents of $mime:
// MIME-Version: 1.0
// Date: Thu, 11 Oct 2018 15:42:25 GMT
// Message-ID: <7CA5D25B2B5E0047A36F2E8CB30362E2>
// Sender: tsales@massmarket.com
// From: tsales@massmarket.com
// To: noreply@4d.com
// To: test@4d.com
// Content-Type: multipart/alternative;
boundary="E0AE5773D5E95245BBBD80DD0687E218"
// Subject: Terrific Sale! This week only!
//
// --E0AE5773D5E95245BBBD80DD0687E218
// Content-Type: text/plain; charset="UTF-8"
// Content-Transfer-Encoding: quoted-printable
//
// Text format email
// --E0AE5773D5E95245BBBD80DD0687E218
// Content-Type: text/html; charset="UTF-8"
// Content-Transfer-Encoding: quoted-printable
//
// <html><body>HTML format email</body></html>
// --E0AE5773D5E95245BBBD80DD0687E218--
```

# Entity

An [entity](#) is an instance of a [Dataclass](#), like a record of the table matching the dataclass in its associated datastore. It contains the same attributes as the dataclass as well as the data values and specific properties and functions.

## Summary

[\*\*.attributeName : any\*\*](#) stores the attribute value for the entity  
[\*\*.clone\(\) : 4D.Entity\*\*](#) creates in memory a new entity referencing the same record as the original entity  
[\*\*.diff\( entityToCompare : 4D.Entity { ; attributesToCompare : Collection } \) : Collection\*\*](#) compares the contents of two entities and returns their differences  
[\*\*.drop\( {mode : Integer} \) : Object\*\*](#) deletes the data contained in the entity from the datastore  
[\*\*.first\(\) : 4D.Entity\*\*](#) returns a reference to the entity in first position of the entity selection which the entity belongs to  
[\*\*.fromObject\( filler : Object \)\*\*](#) fills an entity with the *filler* content  
[\*\*.getDataClass\(\) : 4D.DataClass\*\*](#) returns the dataclass of the entity  
[\*\*.getKey\( { mode : Integer } \) : Text\*\*](#)  
[\*\*.getKey\( { mode : Integer } \) : Integer\*\*](#) returns the primary key value of the entity  
[\*\*.getRemoteContextAttributes\(\) : Text\*\*](#) returns information about the optimization context used by the entity  
[\*\*.getSelection\(\) : 4D.EntitySelection\*\*](#) returns the entity selection which the entity belongs to  
[\*\*.getStamp\(\) : Integer\*\*](#) returns the current value of the stamp of the entity  
[\*\*.indexOf\( { entitySelection : 4D.EntitySelection } \) : Integer\*\*](#) returns the position of the entity in an entity selection  
[\*\*.isNew\(\) : Boolean\*\*](#) returns True if the entity to which it is applied has just been created and has not yet been saved in the datastore  
[\*\*.last\(\) : 4D.Entity\*\*](#) returns a reference to the entity in last position of the entity selection which the entity belongs to  
[\*\*.lock\( { mode : Integer } \) : Object\*\*](#) puts a pessimistic lock on the record referenced by the entity  
[\*\*.next\(\) : 4D.Entity\*\*](#) returns a reference to the next entity in the entity selection which the entity belongs to  
[\*\*.previous\(\) : 4D.Entity\*\*](#) returns a reference to the previous entity in the entity selection which the entity belongs to  
[\*\*.reload\(\) : Object\*\*](#) reloads the content of the entity in memory  
[\*\*.save\( { mode : Integer } \) : Object\*\*](#) saves the changes made to the entity  
[\*\*.toObject\(\) : Object\*\*](#)  
[\*\*.toObject\( filterString : Text { ; options : Integer } \) : Object\*\*](#)  
[\*\*.toObject\( filterCol : Collection { ; options : Integer } \) : Object\*\*](#) returns an object which has been built from the entity  
[\*\*.touched\(\) : Boolean\*\*](#) tests whether or not an entity attribute has been modified since the entity was loaded into memory or saved  
[\*\*.touchedAttributes\(\) : Collection\*\*](#) returns the names of the attributes that have been modified since the entity was loaded into memory  
[\*\*.unlock\(\) : Object\*\*](#) removes the pessimistic lock on the record matching the entity

## **.attributeName**

- History

**.attributeName** : any

## Description

Any dataclass attribute is available as a property of an entity, which stores the attribute value for the entity.

Dataclass attributes can also be reached using the alternate syntax with [ ].

The attribute value type depends on the attribute [kind](#) (relation or storage):

- If *attributeName* kind is **storage**: `.attributeName` returns a value of the same type as *attributeName*.
- If *attributeName* kind is **relatedEntity**: `.attributeName` returns the related entity. Values of the related entity are directly available through cascading properties, for example "myEntity.employer.employees[0].lastname".
- If *attributeName* kind is **relatedEntities**: `.attributeName` returns a new entity selection of related entities. Duplications are removed (an unordered entity selection is returned).

## Example

```
var $myEntity : cs.EmployeeEntity
$myEntity:=ds.Employee.new() //Create a new entity
$myEntity.name:="Dupont" // assign 'Dupont' to the 'name' attribute
$myEntity.firstname:="John" //assign 'John' to the 'firstname'
attribute
$myEntity.save() //save the entity
```

## .clone()

- History

**.clone()** : 4D.Entity

| Parameter | Type         | Description                       |
|-----------|--------------|-----------------------------------|
| Result    | 4D.Entity <- | New entity referencing the record |

## Description

The `.clone()` function creates in memory a new entity referencing the same record as the original entity. This function allows you to update entities separately.

Keep in mind that any modifications done to entities will be saved in the referenced record only when the [.save\(\)](#) function is executed.

This function can only be used with entities already saved in the database. It cannot be called on a newly created entity (for which [.isNew\(\)](#) returns **True**).

## Example

```

var $emp; $empCloned : cs.EmployeeEntity
$emp:=ds.Employee.get(672)
$empCloned:=$emp.clone()

$emp.lastName:="Smith" //Updates done on $emp are not done on
$empCloned

```

## .diff()

- History

**.diff( entityToCompare : 4D.Entity { ; attributesToCompare : Collection } ) : Collection**

| Parameter           | Type          | Description                                    |
|---------------------|---------------|--|
| entityToCompare     | 4D.Entity ->  | Entity to be compared with the original entity |
| attributesToCompare | Collection -> | Name of attributes to be compared              |
| Result              | Collection <- | Differences between the entities               |

### Description

The `.diff()` function compares the contents of two entities and returns their differences.

In *entityToCompare*, pass the entity to be compared to the original entity.

In *attributesToCompare*, you can designate specific attributes to compare. If provided, the comparison is done only on the specified attributes. If not provided, all differences between the entities are returned.

The differences are returned as a collection of objects whose properties are:

| Property name | Type                            | Description                                      |
|---------------|---------------------------------|--|
| attributeName | String                          | Name of the attribute                            |
| value         | any - Depends on attribute type | Value of the attribute in the entity             |
| otherValue    | any - Depends on attribute type | Value of the attribute in <i>entityToCompare</i> |

Only attributes with different values are included in the collection. If no differences are found, `.diff()` returns an empty collection.

The function applies for properties whose [kind](#) is **storage** or **relatedEntity**. In case a related entity has been updated (meaning the foreign key), the name of the related entity and its primary key name are returned as *attributeName* properties (*value* and *otherValue* are empty for the related entity name).

If one of the compared entities is **Null**, an error is raised.

### Example 1

```

var $diff1; $diff2 : Collection
employee:=ds.Employee.query("ID=1001").first()
$clone:=employee.clone()
employee.firstName:="MARIE"
employee.lastName:="SOPHIE"
employee.salary:=500
$diff1:=$clone.diff(employee) // All differences are returned
$diff2:=$clone.diff(employee;New collection("firstName";"lastName"))
// Only differences on firstName and lastName are returned

```

**\$diff1:**

```

[
  {
    "attributeName": "firstName",
    "value": "Natasha",
    "otherValue": "MARIE"
  },
  {
    "attributeName": "lastName",
    "value": "Locke",
    "otherValue": "SOPHIE"
  },
  {
    "attributeName": "salary",
    "value": 66600,
    "otherValue": 500
  }
]

```

**\$diff2:**

```

[
  {
    "attributeName": "firstName",
    "value": "Natasha",
    "otherValue": "MARIE"
  },
  {
    "attributeName": "lastName",
    "value": "Locke",
    "otherValue": "SOPHIE"
  }
]
```

## Example 2

```

var vCompareResult1; vCompareResult2; vCompareResult3;
$attributesToInspect : Collection
vCompareResult1:=New collection
vCompareResult2:=New collection
vCompareResult3:=New collection
$attributesToInspect:=New collection

$e1:=ds.Employee.get(636)
$e2:=ds.Employee.get(636)

$e1.firstName:=$e1.firstName+" update"
$e1.lastName:=$e1.lastName+" update"

$c:=ds.Company.get(117)
$e1.employer:=$c
$e2.salary:=100

$attributesToInspect.push("firstName")
$attributesToInspect.push("lastName")

vCompareResult1:=$e1.diff($e2)
vCompareResult2:=$e1.diff($e2;$attributesToInspect)
vCompareResult3:=$e1.diff($e2;$e1.touchedAttributes())

```

**vCompareResult1 (all differences are returned):**

```

[
  {
    "attributeName": "firstName",
    "value": "Karla update",
    "otherValue": "Karla"
  },
  {
    "attributeName": "lastName",
    "value": "Marrero update",
    "otherValue": "Marrero"
  },
  {
    "attributeName": "salary",
    "value": 33500,
    "otherValue": 100
  },
  {
    "attributeName": "employerID",
    "value": 117,
    "otherValue": 118
  },
  {
    "attributeName": "employer",
    "value": "[object Entity]",// Entity 117 from Company
    "otherValue": "[object Entity]"// Entity 118 from Company
  }
]
```

**vCompareResult2 (only differences on \$attributesToInspect are returned)**

```
[
  {
    "attributeName": "firstName",
    "value": "Karla update",
    "otherValue": "Karla"
  },
  {
    "attributeName": "lastName",
    "value": "Marrero update",
    "otherValue": "Marrero"
  }
]
```

vCompareResult3 (only differences on \$e1 touched attributes are returned)

```
[
  {
    "attributeName": "firstName",
    "value": "Karla update",
    "otherValue": "Karla"
  },
  {
    "attributeName": "lastName",
    "value": "Marrero update",
    "otherValue": "Marrero"
  },
  {
    "attributeName": "employerID",
    "value": 117,
    "otherValue": 118
  },
  {
    "attributeName": "employer",
    "value": "[object Entity]">// Entity 117 from Company
    "otherValue": "[object Entity]"// Entity 118 from Company
  }
]
```

## .drop()

- History

**.drop( {mode : Integer} ) : Object**

| Parameter | Type  | Description                                   |
|-----------|---|---|
| mode      | Integer-> <small>dk force drop if stamp changed</small> | Forces the drop even if the stamp has changed |
| Result    | Object <-Result of drop operation                       |   |

### Description

The `.drop()` function deletes the data contained in the entity from the datastore, from the table related to its Dataclass. Note that the entity remains in memory.

In a multi-user or multi-process application, the `.drop()` function is executed under an "[optimistic lock](#)" mechanism, wherein an internal locking stamp is automatically incremented each time the record is saved.

By default, if the *mode* parameter is omitted, the function will return an error (see below) if the same entity was modified (i.e. the stamp has changed) by another process or user in the meantime.

Otherwise, you can pass the `dk force drop if stamp changed` option in the `mode` parameter: in this case, the entity is dropped even if the stamp has changed (and the primary key is still the same).

## Result

The object returned by `.drop( )` contains the following properties:

| Property       | Type                        | Description  |
|----------------|-----------------------------|--|
| success        | boolean                     | true if the drop action is successful, false otherwise.  |
|                |                             | <b>Available only in case of error:</b>  |
| status(*)      | number                      | Error code, see below  |
| statusText(*)  | text                        | Description of the error, see below  |
|                |                             | <b>Available only in case of pessimistic lock error:</b>   |
| LockKindText   | text                        | "Locked by record"   |
| lockInfo       | object                      | Information about the lock origin  |
| task_id        | number                      | Process id   |
| user_name      | text                        | Session user name on the machine   |
| user4d_alias   | text                        | User alias if defined by <code>SET USER ALIAS</code> , otherwise user name in the 4D directory                         |
| host_name      | text                        | Machine name   |
| task_name      | text                        | Process name   |
| client_version | text                        |  |
|                |                             | <b>Available only in case of serious error (serious error can be trying to duplicate a primary key, disk full...):</b> |
| errors         | collection<br>of<br>objects |  |
| message        | text                        | Error message  |
| component      | text                        | internal component signature (e.g. "dmbg")   |
| signature      | text                        | stands for the database component)   |
| errCode        | number                      | Error code   |

(\*) The following values can be returned in the `status` and `statusText` properties of `Result` object in case of error:

| Constant                                | Value | Comment  |
|---|-------|--|
| dk status entity does not exist anymore | 5     | The entity no longer exists in the data. This error can occur in the following cases: <ul style="list-style-type: none"><li>• the entity has been dropped (the stamp has changed and the memory space is now free)</li><li>• the entity has been dropped and replaced by another one with another primary key (the stamp has changed and a new entity now uses the memory space). When using <code>entity.drop()</code>, this error can be returned when dk force drop if stamp changed option is used. When using <code>entity.lock()</code>, this error can be returned when dk reload if stamp changed option is used</li></ul> <b>Associated statusText:</b> "Entity does not exist anymore" |
| dk status locked                        | 3     | The entity is locked by a pessimistic lock.<br><b>Associated statusText:</b> "Already locked"  |
| dk status serious error                 | 4     | A serious error is a low-level database error (e.g. duplicated key), a hardware error, etc.<br><b>Associated statusText:</b> "Other error"   |
| dk status stamp has changed             | 2     | The internal stamp value of the entity does not match the one of the entity stored in the data (optimistic lock). <ul style="list-style-type: none"><li>• with <code>.save()</code>: error only if the <code>dk auto merge</code> option is not used</li><li>• with <code>.drop()</code>: error only if the <code>dk force drop if stamp changed</code> option is not used</li><li>• with <code>.lock()</code>: error only if the <code>dk reload if stamp changed</code> option is not used</li></ul> <b>Associated statusText:</b> "Stamp has changed"   |
| dk status wrong permission              | 1     | The current privileges do not allow the drop of the entity.<br><b>Associated statusText:</b> "Permission Error"  |

## Example 1

Example without `dk force drop if stamp changed` option:

```
var $employees : cs.EmployeeSelection
var $employee : cs.EmployeeEntity
var $status : Object
$employees:=ds.Employee.query("lastName=:1";"Smith")
$employee:=$employees.first()
$status:=$employee.drop()
Case of
    :($status.success)
        ALERT("You have dropped "+$employee.firstName+
"+$employee.lastName) //The dropped entity remains in memory
    :($status.status=dk status stamp has changed)
        ALERT($status.statusText)
End case
```

## Example 2

Example with `dk force drop if stamp changed` option:

```

var $employees : cs.EmployeeSelection
var $employee : cs.EmployeeEntity
var $status : Object
$employees:=ds.Employee.query("lastName=:1";"Smith")
$employee:=$employees.first()
$status:=$employee.drop(dk force drop if stamp changed)
Case of
    : ($status.success)
        ALERT("You have dropped "+$employee.firstName+
"$employee.lastName) //The dropped entity remains in memory
    : ($status.status=dk status entity does not exist anymore)
        ALERT($status.statusText)
End case

```

## .first()

- History

**.first():** 4D.Entity

| Parameter | Type        | Description  |
|-----------|-------------|--|
| Result    | 4D.Entity<- | Reference to first entity of an entity selection (Null if not found) |

### Description

The `.first()` function returns a reference to the entity in first position of the entity selection which the entity belongs to.

If the entity does not belong to any existing entity selection (i.e. [.getSelection\( \)](#) returns Null), the function returns a Null value.

### Example

```

var $employees : cs.EmployeeSelection
var $employee; $firstEmployee : cs.EmployeeEntity
$employees:=ds.Employee.query("lastName = :1";"H@") //This entity
selection contains 3 entities
$employee:=$employees[2]
$firstEmployee:=$employee.first() // $firstEmployee is the first entity
of the $employees entity selection

```

## .fromObject()

- History

**.fromObject( *filler* : Object )**

| Parameter | Type     | Description                          |
|-----------|----------|--------------------------------------|
| filler    | Object-> | Object from which to fill the entity |

### Description

The `.fromObject()` function fills an entity with the *filler* content.

This function modifies the original entity.

The mapping between the object and the entity is done on the attribute names:

- If a property of the object does not exist in the dataclass, it is ignored.
- Data types must be equivalent. If there is a type mismatch between the object and dataclass, 4D tries to convert the data whenever possible (see [Converting data types](#)), otherwise the attribute is left untouched.
- The primary key can be given as is or with a "\_\_KEY" property (filled with the primary key value). If it does not already exist in the dataclass, the entity is created with the given value when [.save\(\)](#) is called. If the primary key is not given, the entity is created and the primary key value is assigned with respect to database rules. The auto-increment is only computed if the primary key is null.

*filler* can handle a related entity under the following conditions:

- *filler* contains the foreign key itself, or
- *filler* contains a property object with the same name as the related entity, containing a single property named "\_\_KEY".
- if the related entity does not exist, it is ignored.

## Example

With the following \$o object:

```
{
  "firstName": "Mary",
  "lastName": "Smith",
  "salary": 36500,
  "birthDate": "1958-10-27T00:00:00.000Z",
  "woman": true,
  "managerID": 411, // relatedEntity given with PK
  "employerID": 20 // relatedEntity given with PK
}
```

The following code will create an entity with manager and employer related entities.

```
var $o : Object
var $entity : cs.EmpEntity
$entity:=ds.Emp.new()
$entity.fromObject($o)
$entity.save()
```

You could also use a related entity given as an object:

```
{
  "firstName": "Marie",
  "lastName": "Lechat",
  "salary": 68400,
  "birthDate": "1971-09-03T00:00:00.000Z",
  "woman": false,
  "employer": { // relatedEntity given as an object
    "__KEY": "21"
  },
  "manager": { // relatedEntity given as an object
    "__KEY": "411"
  }
}
```

## .getDataClass()

- History

**.getDataClass() : 4D.DataClass**

| Parameter | Type            | Description                                  |
|-----------|-----------------|--|
| Result    | 4D.DataClass <- | DataClass object to which the entity belongs |

## Description

The `.getDataClass()` function returns the dataclass of the entity. This function is useful when writing generic code.

## Example

The following generic code duplicates any entity:

```
//duplicate_entity method
//duplicate_entity($entity)

#DECLARE ($entity : 4D.Entity)
var $entityNew : 4D.Entity
var $status : Object

$entityNew:=$entity.getDataClass().new() //create a new entity in the
parent dataclass
$entityNew.fromObject($entity.toObject()) //get all attributes
$entityNew[$entity.getDataClass().getInfo().primaryKey]:=Null //reset
the primary key
$status:=$entityNew.save() //save the duplicated entity
```

## .getKey()

- History

`.getKey( { mode : Integer } )` : Text  
`.getKey( { mode : Integer } )` : Integer

| Parameter | Type  | Description   |
|-----------|---|---|
| mode      | Integer-> <code>dk key as string</code>                 | primary key is returned as a string, no matter the primary key type |
| Result    | Text <-Value of the text primary key of the entity      |   |
| Result    | Integer<-Value of the numeric primary key of the entity |   |

## Description

The `.getKey()` function returns the primary key value of the entity.

Primary keys can be numbers (Integer) or strings. You can "force" the returned primary key value to be a string, no matter the actual primary key type, by passing the `dk key as string` option in the *mode* parameter.

## Example

```
var $employees : cs.EmployeeSelection
var $employee : cs.EmployeeEntity
$employees:=ds.Employee.query("lastName=:1";"Smith")
$employee:=$employees[0]
ALERT("The primary key is "+$employee.getKey(dk key as string))
```

## .getRemoteContextAttributes()

- History

## .getRemoteContextAttributes() : Text

| Parameter | Type  | Description |
|-----------|---|-------------|
| result    | Text <- Context attributes linked to the entity, separated by a comma |             |

**Advanced mode:** This function is intended for developers who need to customize ORDA default features for specific configurations. In most cases, you will not need to use it.

### Description

The `.getRemoteContextAttributes()` function returns information about the optimization context used by the entity .

If there is no [optimization context](#) for the entity, the function returns an empty Text.

### Example

```
var $ds : 4D.DataStoreImplementation
var $address : cs.AddressEntity
var $p : cs.PersonsEntity
var $contextA : Object
var $info : Text
var $text : Text

$ds:=Open datastore(New object("hostname"; "www.myserver.com"); "myDS")

$contextA:=New object("context"; "contextA")

$address:=$ds.Address.get(1; $contextA)
$text:=""
For each ($p; $address.persons)
    $text:=$p.firstname+" "+$p.lastname
End for each

$info:=$address.getRemoteContextAttributes()

// $info = "persons,persons.lastname,persons.firstname"
```

### See also

[EntitySelection.getRemoteContextAttributes\(\)](#)  
[.clearAllRemoteContexts\(\)](#)  
[.getRemoteContextInfo\(\)](#)  
[.getAllRemoteContexts\(\)](#)  
[.setRemoteContextInfo\(\)](#)

## .getSelection()

- History

**.getSelection():** 4D.EntitySelection

| Parameter | Type                  | Description  |
|-----------|-----------------------|--|
| Result    | 4D.EntitySelection <- | Entity selection to which the entity belongs (Null if not found) |

### Description

The `.getSelection()` function returns the entity selection which the entity belongs to.

If the entity does not belong to an entity selection, the function returns Null.

## Example

```
var $emp : cs.EmployeeEntity
var $employees; $employees2 : cs.EmployeeSelection
$emp:=ds.Employee.get(672) // This entity does not belong to any
entity selection
$employees:=$emp.getSelection() // $employees is Null

$employees2:=ds.Employee.query("lastName=:1";"Smith") //This entity
selection contains 6 entities
$emp:=$employees2[0] // This entity belongs to an entity selection

ALERT("The entity selection contains
"+String($emp.getSelection().length)+" entities")
```

## .getStamp()

- History

**.getStamp()** : Integer

| Parameter | Type      | Description   |
|-----------|-----------|---|
| Result    | Integer<- | Stamp of the entity (0 if entity has just been created) |

## Description

The `.getStamp()` function returns the current value of the stamp of the entity.

The internal stamp is automatically incremented by 4D each time the entity is saved. It manages concurrent user access and modifications to the same entities (see [Entity locking](#)).

For a new entity (never saved), the function returns 0. To know if an entity has just been created, it is recommended to use [.isNew\(\)](#).

## Example

```
var $entity : cs.EmployeeEntity
var $stamp : Integer

$entity:=ds.Employee.new()
$entity.lastname:="Smith"
$entity.save()
$stamp:=$entity.getStamp() // $stamp=1

$entity.lastname:="Wesson"
$entity.save()
$stamp:=$entity.getStamp() // $stamp=2
```

## .indexOf()

- History

**.indexOf( { entitySelection : 4D.EntitySelection } )** : Integer

| Parameter       | Type                  | Description  |
|-----------------|-----------------------|--|
| entitySelection | 4D.EntitySelection -> | Position of the entity is given according to this entity selection |
| Result          | Integer               | <- Position of the entity in an entity selection                   |

## Description

The `.indexOf()` function returns the position of the entity in an entity selection.

By default if the `entitySelection` parameter is omitted, the function returns the entity's position within its own entity selection. Otherwise, it returns the position of the entity within the specified `entitySelection`.

The resulting value is included between 0 and the length of the entity selection -1.

- If the entity does not have an entity selection or does not belong to `entitySelection`, the function returns -1.
- If `entitySelection` is Null or does not belong to the same dataclass as the entity, an error is raised.

## Example

```
var $employees : cs.EmployeeSelection
var $employee : cs.EmployeeEntity
$employees:=ds.Employee.query("lastName = :1";"H@") //This entity
selection contains 3 entities
$employee:=$employees[1] //This entity belongs to an entity selection
ALERT("The index of the entity in its own entity selection is
"+String($employee.indexOf())) //1

$employee:=ds.Employee.get(725) //This entity does not belong to an
entity selection
ALERT("The index of the entity is "+String($employee.indexOf())) // -1
```

## .isNew()

- History

`.isNew()` : Boolean

| Parameter | Type       | Description  |
|-----------|------------|--|
| Result    | Boolean <- | True if entity has just been created and not yet saved.<br>Otherwise, False. |

## Description

The `.isNew()` function returns True if the entity to which it is applied has just been created and has not yet been saved in the datastore. Otherwise, it returns False.

## Example

```
var $emp : cs.EmployeeEntity
$emp:=ds.Employee.new()
If ($emp.isNew())
    ALERT("This is a new entity")
End if
```

## .last()

- History

**.last()** : 4D.Entity

| Parameter | Type         | Description   |
|-----------|--------------|---|
| Result    | 4D.Entity <- | Reference to last entity of an entity selection (Null if not found) |

### Description

The `.last()` function returns a reference to the entity in last position of the entity selection which the entity belongs to.

If the entity does not belong to any existing entity selection (i.e. [.getSelection\(\)](#) returns Null), the function returns a Null value.

## Example

```
var $employees : cs.EmployeeSelection
var $employee; $lastEmployee : cs.EmployeeEntity
$employees:=ds.Employee.query("lastName = :1";"H@") //This entity
selection contains 3 entities
$employee:=$employees[0]
$lastEmployee:=$employee.last() // $lastEmployee is the last entity of
the $employees entity selection
```

## .lock()

- History

**.lock( { mode : Integer } )** : Object

| Parameter | Type  | Description                            |
|-----------|---|--|
| mode      | Integer-> <code>dk reload if stamp changed</code> | Reload before locking if stamp changed |
| Result    | Object <-   | Result of lock operation               |

### Description

The `.lock()` function puts a pessimistic lock on the record referenced by the entity. The [lock is set](#) for a record and all the references of the entity in the current process.

Other processes will see this record as locked (the `result.success` property will contain False if they try to lock the same entity using this function). Only functions executed in the "locking" session are allowed to edit and save the attributes of the entity. The entity can be loaded as read-only by other sessions, but they will not be able to enter and save values.

A record locked by `.lock()` is unlocked:

- when the [unlock\(\)](#) function is called on a matching entity in the same process
- automatically, when it is no longer referenced by any entities in memory. For example, if the lock is put only on one local reference of an entity, the entity is unlocked when the function ends. As long as there are references to the entity in memory, the record remains locked.

An entity can also be [locked by a REST session](#), in which case it can only be unlocked by the session.

By default, if the *mode* parameter is omitted, the function will return an error (see below) if the same entity was modified (i.e. the stamp has changed) by another process or user in the meantime.

Otherwise, you can pass the `dk reload if stamp changed` option in the *mode* parameter: in this case, no error is returned and the entity is reloaded when the stamp has changed (if the entity still exists and the primary key is still the same).

## Result

The object returned by `.lock( )` contains the following properties:

| <b>Property</b>    | <b>Type</b>                 | <b>Description</b>  |  |
|--------------------|-----------------------------|---|--|
| success            | boolean                     | true if the lock action is successful (or if the entity is already locked in the current process), false otherwise.<br><br><b>Available only if <code>dk reload if stamp changed</code> option is used:</b> |  |
| <b>wasReloaded</b> | boolean                     | true if the entity was reloaded with success, false otherwise.<br><br><b>Available only in case of error:</b>   |  |
| status(*)          | number                      | Error code, see below   |  |
| statusText(*)      | text                        | Description of the error, see below<br><br><b>Available only in case of pessimistic lock error:</b>   |  |
| lockKindText       | text                        | "Locked by record" if locked by a 4D process, "Locked by session" if locked by a REST session   |  |
| lockInfo           | object                      | Information about the lock origin. Returned properties depend on the lock origin (4D process or REST session).<br><br><b>Available only for a 4D process lock:</b>  |  |
|                    | task_id                     | number  | Process ID   |
|                    | user_name                   | text  | Session user name on the machine   |
|                    | user4d_alias                | text  | Name or alias of the 4D user   |
|                    | user4d_id                   | number  | User id in the 4D database directory   |
|                    | host_name                   | text  | Machine name   |
|                    | task_name                   | text  | Process name   |
|                    | client_version              | text  | Version of the client  |
|                    | host                        | text  | <b>Available only for a REST session lock:</b><br>URL that locked the entity (e.g. " <a href="http://www.myserver.com">www.myserver.com</a> ")   |
|                    | IPAddr                      | text  | IP address of the locker (e.g. "127.0.0.1")<br>userAgent of the locker (e.g. Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36") |
|                    | userAgent                   | text  | <b>Available only in case of serious error</b><br>(primary key already exists, disk full...):  |
| errors             | collection<br>of<br>objects |   |  |
|                    | message                     | text  | Error message  |
|                    | component<br>signature      | text  | internal component signature (e.g. "dmbg" stands for the database component)   |
|                    | errCode                     | number  | Error code   |

(\*) The following values can be returned in the *status* and *statusText* properties of the *Result* object in case of error:

| Constant   | Value | Comment  |
|--|-------|--|
| dk<br>status<br>entity<br>does not<br>exist<br>anymore | 5     | <p>The entity no longer exists in the data. This error can occur in the following cases:</p> <ul style="list-style-type: none"> <li>• the entity has been dropped (the stamp has changed and the memory space is now free)</li> <li>• the entity has been dropped and replaced by another one with another primary key (the stamp has changed and a new entity now uses the memory space). When using <code>.drop()</code>, this error can be returned when dk force drop if stamp changed option is used. When using <code>.lock()</code>, this error can be returned when dk reload if stamp changed option is used</li> </ul> |

**Associated statusText:** "Entity does not exist anymore"

|   |   |   |
|---|---|---|
| dk<br>status<br>locked                  | 3 | The entity is locked by a pessimistic lock. <b>Associated statusText:</b> "Already locked"  |
| dk<br>status<br>serious<br>error        | 4 | A serious error is a low-level database error (e.g. duplicated key), a hardware error, etc. <b>Associated statusText:</b> "Other error"   |
| dk<br>status<br>stamp<br>has<br>changed | 2 | <p>The internal stamp value of the entity does not match the one of the entity stored in the data (optimistic lock).</p> <ul style="list-style-type: none"> <li>• with <code>.save()</code>: error only if the <code>dk auto merge</code> option is not used</li> <li>• with <code>.drop()</code>: error only if the <code>dk force drop if stamp changed</code> option is not used</li> <li>• with <code>.lock()</code>: error only if the <code>dk reload if stamp changed</code> option is not used</li> </ul> |

**Associated statusText:** "Stamp has changed"

## Example 1

Example with error:

```
var $employee : cs.EmployeeEntity
var $status : Object
$employee:=ds.Employee.get(716)
$status:=$employee.lock()
Case of
    :($status.success)
        ALERT("You have locked "+$employee.firstName+
"$employee.lastName")
    :($status.status=dk status stamp has changed)
        ALERT($status.statusText)
End case
```

## Example 2

Example with `dk reload if stamp changed` option:

```

var $employee : cs.EmployeeEntity
var $status : Object
$employee:=ds.Employee.get(717)
$status:=$employee.lock(dk reload if stamp changed)
Case of
    : ($status.success)
        ALERT("You have locked "+$employee.firstName+
"$employee.lastName")
    : ($status.status=dk status entity does not exist anymore)
        ALERT($status.statusText)
End case

```

## .next()

- History

**.next() : 4D.Entity**

| Parameter | Type         | Description  |
|-----------|--------------|--|
| Result    | 4D.Entity <- | Reference to next entity in the entity selection (Null if not found) |

### Description

The `.next()` function returns a reference to the next entity in the entity selection which the entity belongs to.

If the entity does not belong to any existing entity selection (i.e. [.getSelection\(\)](#) returns Null), the function returns a Null value.

If there is no valid next entity in the entity selection (i.e. you are on the last entity of the selection), the function returns Null. If the next entity has been dropped, the function returns the next valid entity (and eventually Null).

### Example

```

var $employees : cs.EmployeeSelection
var $employee; $nextEmployee : cs.EmployeeEntity
$employees:=ds.Employee.query("lastName = :1;"H@") //This entity
selection contains 3 entities
$employee:=$employees[0]
$nextEmployee:=$employee.next() // $nextEmployee is the second entity
of the $employees entity selection

```

## .previous()

- History

**.previous() : 4D.Entity**

| Parameter | Type         | Description  |
|-----------|--------------|--|
| Result    | 4D.Entity <- | Reference to previous entity in the entity selection (Null if not found) |

### Description

The `.previous()` function returns a reference to the previous entity in the entity selection which the entity belongs to.

If the entity does not belong to any existing entity selection (i.e. [.getSelection\(\)](#) returns Null), the function returns a Null value.

If there is no valid previous entity in the entity selection (i.e. you are on the first entity of the selection), the function returns Null. If the previous entity has been dropped, the function returns the previous valid entity (and eventually Null).

## Example

```
var $employees : cs.EmployeeSelection
var $employee; $previousEmployee : cs.EmployeeEntity
$employees:=ds.Employee.query("lastName = :1";"H@") //This entity
selection contains 3 entities
$employee:=$employees[1]
$previousEmployee:=$employee.previous() // $previousEmployee is the
first entity of the $employees entity selection
```

## .reload()

- History

### .reload() : Object

| Parameter | Type                  | Description |
|-----------|-----------------------|-------------|
| Result    | Object<-Status object |             |

### Description

The `.reload()` function reloads the content of the entity in memory, according to information stored in the table related to the dataclass in the datastore. The reload is done only if the entity still exists with the same primary key.

### Result

The object returned by `.reload()` contains the following properties:

| Property      | Type    | Description   |
|---------------|---------|---|
| success       | boolean | True if the reload action is successful, False otherwise. <b>Available only in case of error:</b> |
| status(*)     | number  | Error code, see below   |
| statusText(*) | text    | Description of the error, see below   |

(\*) The following values can be returned in the *status* and *statusText* properties of *Result* object in case of error:

| Constant  | Value | Comment   |
|---|-------|---|
| dk<br>status<br>entity<br>does<br>not<br>exist<br>anymore | 5     | The entity no longer exists in the data. This error can occur in the following cases: <ul style="list-style-type: none"> <li>• the entity has been dropped (the stamp has changed and the memory space is now free)</li> <li>• the entity has been dropped and replaced by another one with another primary key (the stamp has changed and a new entity now uses the memory space). When using <code>.drop()</code>, this error can be returned when <code>dk force drop if stamp changed</code> option is used. When using <code>.lock()</code>, this error can be returned when <code>dk reload if stamp changed</code> option is used</li> </ul> |
| dk<br>status<br>serious<br>error                          | 4     | <b>Associated statusText:</b> "Entity does not exist anymore"<br><br>A serious error is a low-level database error (e.g. duplicated key), a hardware error, etc.<br><b>Associated statusText:</b> "Other error"   |

## Example

```

var $employee : cs.EmployeeEntity
var $employees : cs.EmployeeSelection
var $result : Object

$employees:=ds.Employee.query("lastName=:1";"Hollis")
$employee:=$employees[0]
$employee.firstName:="Mary"
$result:=$employee.reload()
Case of
    :($result.success)
        ALERT("Reload has been done")
    :($result.status=dk status entity does not exist anymore)
        ALERT("The entity has been dropped")
End case

```

## .save()

- History

**.save( { mode : Integer } ) : Object**

| Parameter | Type                                 | Description                            |
|-----------|--------------------------------------|--|
| mode      | Integer-> <code>dk auto merge</code> | Mode: Enables the automatic merge mode |
| Result    | Object <-Result of save operation    |  |

## Description

The `.save()` function saves the changes made to the entity in the table related to its dataClass. You must call this method after creating or modifying an entity if you want to save the changes made to it.

The save operation is executed only if at least one entity attribute has been "touched" (see the `.touched()` and `.touchedAttributes()` functions). Otherwise, the function does nothing (the trigger is not called).

In a multi-user or multi-process application, the `.save()` function is executed under an "[optimistic lock](#)" mechanism, wherein an internal locking stamp is automatically

incremented each time the record is saved.

By default, if the `mode` parameter is omitted, the method will return an error (see below) whenever the same entity has been modified by another process or user in the meantime, no matter the modified attribute(s).

Otherwise, you can pass the `dk auto merge` option in the `mode` parameter: when the automatic merge mode is enabled, a modification done concurrently by another process/user on the same entity but on a different attribute will not result in an error. The resulting data saved in the entity will be the combination (the "merge") of all non-concurrent modifications (if modifications were applied to the same attribute, the save fails and an error is returned, even with the auto merge mode).

The automatic merge mode is not available for attributes of Picture, Object, and Text type when stored outside of the record. Concurrent changes in these attributes will result in a `dk status stamp has changed` error.

## Result

The object returned by `.save()` contains the following properties:

| Property           | Type                        | Description   |
|--------------------|-----------------------------|---|
| success            | boolean                     | True if the save action is successful, False otherwise.<br><b>Available only if <code>dk auto merge</code> option is used:</b>  |
| autoMerged         | boolean                     | True if an auto merge was done, False otherwise.<br><b>Available only in case of error:</b>                                     |
| status             | number                      | Error code, <a href="#">see below</a>   |
| statusText         | text                        | Description of the error, <a href="#">see below</a><br><b>Available only in case of pessimistic lock error:</b>                 |
| lockKindText       | text                        | "Locked by record"  |
| lockInfo           | object                      | Information about the lock origin   |
| task_id            | number                      | Process id  |
| user_name          | text                        | Session user name on the machine  |
| user4d_alias       | text                        | User alias if defined by <code>SET USER ALIAS</code> , otherwise user name in the 4D directory                                  |
| host_name          | text                        | Machine name  |
| task_name          | text                        | Process name  |
| client_version     | text                        | <br><b>Available only in case of serious error</b><br>(serious error - can be trying to duplicate a primary key, disk full...): |
| errors             | collection<br>of<br>objects |   |
| message            | text                        | Error message   |
| componentSignature | text                        | Internal component signature (e.g. "dmbg" stands for the database component)  |
| errCode            | number                      | Error code  |

## status and statusText

The following values can be returned in the `status` and `statusText` properties of Result object in case of error:

| <b>Constant Value</b>                                  | <b>Comment</b>   |
|--|--|
| <code>dk status automerge 6 failed</code>              | (Only if the <code>dk auto merge</code> option is used) The automatic merge option failed when saving the entity. <b>Associated statusText:</b> "Auto merge failed"  |
| <code>dk status entity does not exist anymore 5</code> | The entity no longer exists in the data. This error can occur in the following cases: <ul style="list-style-type: none"><li>• the entity has been dropped (the stamp has changed and the memory space is now free)</li><li>• the entity has been dropped and replaced by another one with another primary key (the stamp has changed and a new entity now uses the memory space). When using <code>.drop()</code>, this error can be returned when <code>dk force drop if stamp changed</code> option is used. When using <code>.lock()</code>, this error can be returned when <code>dk reload if stamp changed</code> option is used</li></ul> |
| <code>dk status locked 3</code>                        | <b>Associated statusText:</b> "Entity doesnot exist anymore"   |
| <code>dk status serious error 4</code>                 | The entity is locked by a pessimistic lock. <b>Associated statusText:</b> "Already locked"   |
| <code>dk status stamp has changed 2</code>             | A serious error is a low-level database error (e.g. duplicated key), a hardware error, etc. <b>Associated statusText:</b> "Other error"  |
| <code>dk status wrong permission 1</code>              | The internal stamp value of the entity does not match the one of the entity stored in the data (optimistic lock). <ul style="list-style-type: none"><li>• with <code>.save()</code>: error only if the <code>dk auto merge</code> option is not used</li><li>• with <code>.drop()</code>: error only if the <code>dk force drop if stamp changed</code> option is not used</li><li>• with <code>.lock()</code>: error only if the <code>dk reload if stamp changed</code> option is not used</li></ul> <b>Associated statusText:</b> "Stamp has changed"   |
| <code>dk status wrong permission 1</code>              | The current privileges do not allow the save of the entity.<br><b>Associated statusText:</b> "Permission Error"  |

## Example 1

Creating a new entity:

```
var $status : Object
var $employee : cs.EmployeeEntity
$employee:=ds.Employee.new()
$employee.firstName:="Mary"
$employee.lastName:="Smith"
$status:=$employee.save()
If ($status.success)
    ALERT("Employee created")
End if
```

## Example 2

Updating an entity without `dk auto merge` option:

```

var $status : Object
var $employee : cs.EmployeeEntity
var $employees : cs.EmployeeSelection
$employees:=ds.Employee.query("lastName=:1";"Smith")
$employee:=$employees.first()
$employee.lastName:="Mac Arthur"
$status:=$employee.save()
Case of
    :($status.success)
        ALERT("Employee updated")
    :($status.status=dk status stamp has changed)
        ALERT($status.statusText)
End case

```

### Example 3

Updating an entity with `dk auto merge` option:

```

var $status : Object

var $employee : cs.EmployeeEntity
var $employees : cs.EmployeeSelection

$employees:=ds.Employee.query("lastName=:1";"Smith")
$employee:=$employees.first()
$employee.lastName:="Mac Arthur"
$status:=$employee.save(dk auto merge)
Case of
    :($status.success)
        ALERT("Employee updated")
    :($status.status=dk status automerge failed)
        ALERT($status.statusText)
End case

```

## .toObject()

- History

**.toObject()** : Object  
**.toObject( filterString : Text { ; options : Integer} )** : Object  
**.toObject( filterCol : Collection { ; options : Integer } )** : Object

| Parameter    | Type       | Description   |
|--------------|------------|---|
| filterString | Text       | -> Attribute(s) to extract (comma-separated string)   |
| filterCol    | Collection | -> Collection of attribute(s) to extract  |
| options      | Integer    | -> <code>dk with primary key</code> : adds the __KEY property;<br><code>dk with stamp</code> : adds the _STAMP property |
| Result       | Object     | <- Object built from the entity   |

### Description

The `.toObject()` function returns an object which has been built from the entity. Property names in the object match attribute names of the entity.

If no filter is specified, or if the `filterString` parameter contains an empty string or "\*", the returned object will contain:

- all storage entity attributes
- attributes of the `relatedEntity kind`: you get a property with the same name as the related entity (name of the many-to-one link). Attribute is extracted with the

simple form.

- attributes of the `relatedEntities` [kind](#): attribute is not returned.

In the first parameter, you pass the entity attribute(s) to extract. You can pass:

- `filterString`: a string with property paths separated with commas: "propertyPath1, propertyPath2, ...", or
- `filterCol`: a collection of strings: ["propertyPath1","propertyPath2";...]

If a filter is specified for attributes of the `relatedEntity` [kind](#):

- `propertyPath = "relatedEntity"` -> it is extracted with simple form: an object with property `__KEY` (primary key).
- `propertyPath = "relatedEntity.*"` -> all the properties are extracted
- `propertyPath = "relatedEntity.propertyName1; relatedEntity.propertyName2; ..."` -> only those properties are extracted

If a filter is specified for attributes of the `relatedEntities` [kind](#):

- `propertyPath = "relatedEntities.*"` -> all the properties are extracted
- `propertyPath = "relatedEntities.propertyName1; relatedEntities.propertyName2; ..."` -> only those properties are extracted

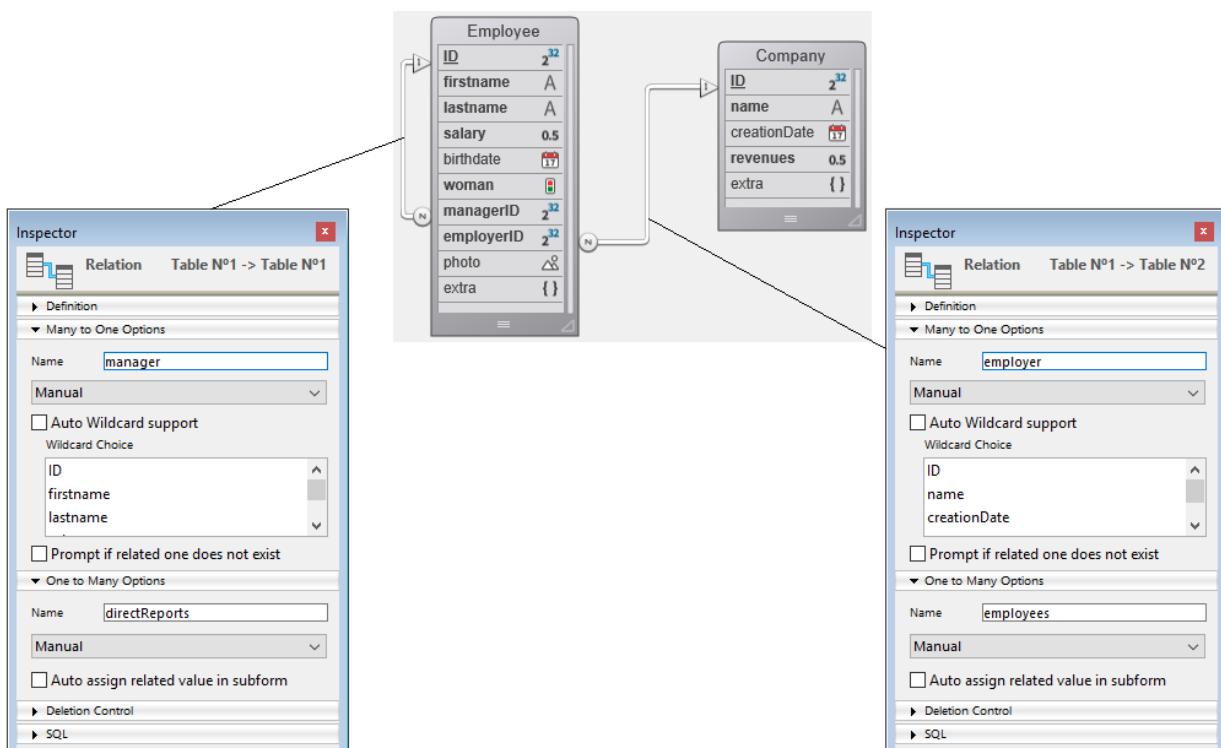
In the `options` parameter, you can pass the `dk with primary key` and/or `dk with stamp` selector(s) to add the entity's primary keys and/or stamps in extracted objects.

## ⚠ WARNING

If you use another attribute than the primary key as the One attribute in a relation, the value of this attribute will be written in the "`__KEY`" property. Keep in mind that it is recommended to use the primary key as One attribute in your relations, especially when you use `.toObject()` and `.fromObject()` functions.

## Example 1

The following structure will be used throughout all examples of this section:



**Without filter parameter:**

```
employeeObject:=employeeSelected.toObject()
```

**Returns:**

```
{
    "ID": 413,
    "firstName": "Greg",
    "lastName": "Wahl",
    "salary": 0,
    "birthDate": "1963-02-01T00:00:00.000Z",
    "woman": false,
    "managerID": 412,
    "employerID": 20,
    "photo": "[object Picture]",
    "extra": null,
    "employer": { // relatedEntity extracted with simple form
        "__KEY": 20
    },
    "manager": {
        "__KEY": 412
    }
}
```

## **Example 2**

Extracting the primary key and the stamp:

```
employeeObject:=employeeSelected.toObject(""); dk with primary key+dk
with stamp)
```

**Returns:**

```
{
    "__KEY": 413,
    "__STAMP": 1,
    "ID": 413,
    "firstName": "Greg",
    "lastName": "Wahl",
    "salary": 0,
    "birthDate": "1963-02-01T00:00:00.000Z",
    "woman": false,
    "managerID": 412,
    "employerID": 20,
    "photo": "[object Picture]",
    "extra": null,
    "employer": {
        "__KEY": 20
    },
    "manager": {
        "__KEY": 412
    }
}
```

## **Example 3**

Expanding all the properties of `relatedEntities`:

```
employeeObject:=employeeSelected.toObject("directReports.*")
```

```

{
  "directReports": [
    {
      "ID": 418,
      "firstName": "Lorena",
      "lastName": "Boothe",
      "salary": 44800,
      "birthDate": "1970-10-02T00:00:00.000Z",
      "woman": true,
      "managerID": 413,
      "employerID": 20,
      "photo": "[object Picture]",
      "extra": null,
      "employer": {
        "__KEY": 20
      },
      "manager": {
        "__KEY": 413
      }
    },
    {
      "ID": 419,
      "firstName": "Drew",
      "lastName": "Caudill",
      "salary": 41000,
      "birthDate": "2030-01-12T00:00:00.000Z",
      "woman": false,
      "managerID": 413,
      "employerID": 20,
      "photo": "[object Picture]",
      "extra": null,
      "employer": {
        "__KEY": 20
      },
      "manager": {
        "__KEY": 413
      }
    },
    {
      "ID": 420,
      "firstName": "Nathan",
      "lastName": "Gomes",
      "salary": 46300,
      "birthDate": "2010-05-29T00:00:00.000Z",
      "woman": false,
      "managerID": 413,
      "employerID": 20,
      "photo": "[object Picture]",
      "extra": null,
      "employer": {
        "__KEY": 20
      },
      "manager": {
        "__KEY": 413
      }
    }
  ]
}

```

#### **Example 4**

Extracting some properties of `relatedEntities`:

```
employeeObject:=employeeSelected.toObject("firstName,  
directReports.lastName")
```

Returns:

```
{  
    "firstName": "Greg",  
    "directReports": [  
        {  
            "lastName": "Boothe"  
        },  
        {  
            "lastName": "Caudill"  
        },  
        {  
            "lastName": "Gomes"  
        }  
    ]  
}
```

## Example 5

Extracting a `relatedEntity` with simple form:

```
$coll:=New collection("firstName";"employer")  
employeeObject:=employeeSelected.toObject($coll)
```

Returns:

```
{  
    "firstName": "Greg",  
    "employer": {  
        "__KEY": 20  
    }  
}
```

## Example 6

Extracting all the properties of a `relatedEntity`:

```
employeeObject:=employeeSelected.toObject("employer.*")
```

Returns:

```
{  
    "employer": {  
        "ID": 20,  
        "name": "India Astral Secretary",  
        "creationDate": "1984-08-25T00:00:00.000Z",  
        "revenues": 12000000,  
        "extra": null  
    }  
}
```

## Example 7

Extracting some properties of a `relatedEntity`:

```
$col:=New collection  
$col.push("employer.name")  
$col.push("employer.revenues")  
employeeObject:=employeeSelected.toObject($col)
```

Returns:

```
{  
    "employer": {  
        "name": "India Astral Secretary",  
        "revenues": 12000000  
    }  
}
```

## .touched( )

- History

**.touched()** : Boolean

| Parameter | Type       | Description   |
|-----------|------------|---|
| Result    | Boolean <- | True if at least one entity attribute has been modified and not yet saved, else False |

### Description

The `.touched()` function tests whether or not an entity attribute has been modified since the entity was loaded into memory or saved.

If an attribute has been modified or calculated, the function returns True, else it returns False. You can use this function to determine if you need to save the entity.

This function returns False for a new entity that has just been created (with `.new()`). Note however that if you use a function which calculates an attribute of the entity, the `.touched()` function will then return True. For example, if you call `.getKey()` to calculate the primary key, `.touched()` returns True.

### Example

In this example, we check to see if it is necessary to save the entity:

```
var $emp : cs.EmployeeEntity  
$emp:=ds.Employee.get(672)  
$emp.firstName:=$emp.firstName //Even if updated with the same value,  
the attribute is marked as touched  
  
If($emp.touched()) //if at least one of the attributes has been  
changed  
    $emp.save()  
End if // otherwise, no need to save the entity
```

## .touchedAttributes( )

- History

**.touchedAttributes()** : Collection

| Parameter | Type          | Description                                      |
|-----------|---------------|--|
| Result    | Collection <- | Names of touched attributes, or empty collection |

### Description

The `.touchedAttributes()` function returns the names of the attributes that have been modified since the entity was loaded into memory.

This applies for attributes of the `kind` `storage` or `relatedEntity`.

In the case of a related entity having been touched (i.e., the foreign key), the name of the related entity and its primary key's name are returned.

If no entity attribute has been touched, the method returns an empty collection.

### Example 1

```
var $touchedAttributes : Collection
var $emp : cs.EmployeeEntity

$touchedAttributes:=New collection
$emp:=ds.Employee.get(725)
$emp.firstName:=$emp.firstName //Even if updated with the same value,
the attribute is marked as touched
$emp.lastName:="Martin"
$touchedAttributes:=$emp.touchedAttributes()
//$touchedAttributes: ["firstName", "lastName"]
```

### Example 2

```
var $touchedAttributes : Collection
var $emp : cs.EmployeeEntity
var $company : cs.CompanyEntity

$touchedAttributes:=New collection

$emp:=ds.Employee.get(672)
$emp.firstName:=$emp.firstName
$emp.lastName:="Martin"

$company:=ds.Company.get(121)
$emp.employer:=$company

$touchedAttributes:=$emp.touchedAttributes()

//collection $touchedAttributes:
["firstName", "lastName", "employer", "employerID"]
```

In this case:

- `firstName` and `lastName` have a `storage` kind
- `employer` has a `relatedEntity` kind
- `employerID` is the foreign key of the employer related entity

## .unlock()

- History

### .unlock() : Object

#### Parameter Type Description

Result      Object <- Status object

#### Description

The `.unlock()` function removes the pessimistic lock on the record matching the entity in the datastore and table related to its dataclass.

For more information, please refer to [Entity locking](#) section.

A record is automatically unlocked when it is no longer referenced by any entities in the locking process (for example: if the lock is put only on one local reference of an entity, the entity and thus the record is unlocked when the process ends).

When a record is locked, it must be unlocked from the locking process and on the entity reference which put the lock. For example:

```
$e1:=ds.Emp.all()[0]
$e2:=ds.Emp.all()[0]
$res:=$e1.lock() // $res.success=true
$res:=$e2.unlock() // $res.success=false
$res:=$e1.unlock() // $res.success=true
```

## Result

The object returned by `.unlock()` contains the following property:

| Property | Type    | Description  |
|----------|---------|--|
| success  | Boolean | True if the unlock action is successful, False otherwise. If the unlock success is done on a dropped entity, on a non locked record, or on a record locked by another process or entity, success is False. |

## Example

```
var $employee : cs.EmployeeEntity
var $status : Object

$employee:=ds.Employee.get(725)
$status:=$employee.lock()
... //processing
$status:=$employee.unlock()
If($status.success)
    ALERT("The entity is now unlocked")
End if
```

# EntitySelection

An entity selection is an object containing one or more reference(s) to [entities](#) belonging to the same [Dataclass](#). An entity selection can contain 0, 1 or X entities from the dataclass -- where X can represent the total number of entities contained in the dataclass.

Entity selections can be created from existing selections using various functions of the [DataClass class](#) such as [.all\(\)](#) or [.query\(\)](#), or functions of the [EntityClass class](#) itself, such as [.and\(\)](#) or [orderBy\(\)](#). You can also create blank entity selections using the [dataClass.newSelection\(\)](#) function or the [Create new selection](#) command.

## Summary

[\[Index\] : 4D.Entity](#) allows you to access entities within the entity selection using the standard collection syntax

[.attributeName : Collection](#)

[.attributeName : 4D.EntitySelection](#) a "projection" of values for the attribute in the entity selection

[.add\( entity : 4D.Entity \) : 4D.EntitySelection](#)

[.add\( entitySelection : 4D.EntitySelection \) : 4D.EntitySelection](#) adds the specified *entity* or *entitySelection* to the original entity selection and returns the modified entity selection

[.and\( entity : 4D.Entity \) : 4D.EntitySelection](#)

[.and\( entitySelection : 4D.EntitySelection \) : 4D.EntitySelection](#) combines the entity selection with an *entity* or *entitySelection* parameter using the logical AND operator

[.average\( attributePath : Text \) : Real](#) returns the arithmetic mean (average) of all the non-null values of *attributePath* in the entity selection

[.contains\( entity : 4D.Entity \) : Boolean](#) returns true if entity reference belongs to the entity selection

[.copy\( { option : Integer } \) : 4D.EntitySelection](#) returns a copy of the original entity selection

[.count\( attributePath : Text \) : Real](#) returns the number of entities in the entity selection with a non-null value in *attributePath*

[.distinct\( attributePath : Text { ; options : Integer } \) : Collection](#) returns a collection containing only distinct (different) values from the *attributePath* in the entity selection

[.distinctPaths\( attribute : Text \) : Collection](#) returns a collection of distinct paths found in the indexed object *attribute* for the entity selection

[.drop\( { mode : Integer } \) : 4D.EntitySelection](#) removes the entities belonging to the entity selection from the table related to its dataclass within the datastore

[.extract\( attributePath : Text { ; option : Integer } \) : Collection](#)

[.extract\( attributePath { ; targetPath } { ; ...attributePathN : Text ; targetPathN : Text } \) : Collection](#) returns a collection containing *attributePath* values extracted from the entity selection

[.first\(\) : 4D.Entity](#) returns a reference to the entity in the first position of the entity selection

[.getDataClass\(\) : 4D.DataClass](#) returns the dataclass of the entity selection

[.getRemoteContextAttributes\(\) : Text](#) returns information about the optimization context used by the entity

[.isAlterable\(\) : Boolean](#) returns True if the entity selection is alterable

[.isOrdered\(\) : Boolean](#) returns True if the entity selection is ordered

**.last() : 4D.Entity** returns a reference to the entity in last position of the entity selection

**.length : Integer** returns the number of entities in the entity selection

**.max( attributePath : Text ) : any** returns the highest (or maximum) value among all the values of *attributePath* in the entity selection

**.min( attributePath : Text ) : any** returns the lowest (or minimum) value among all the values of *attributePath* in the entity selection

**.minus( entity : 4D.Entity { ; keepOrder : Integer } ) : 4D.EntitySelection**

**.minus( entitySelection : 4D.EntitySelection { ; keepOrder : Integer } ) : 4D.EntitySelection** excludes from the entity selection to which it is applied the *entity* or the entities of *entitySelection* and returns the resulting entity selection

**.or( entity : 4D.Entity ) : 4D.EntitySelection**

**.or( entitySelection : 4D.EntitySelection ) : 4D.EntitySelection** combines the entity selection with the *entity* or *entitySelection* parameter using the logical (not exclusive) OR operator

**.orderBy( pathString : Text ) : 4D.EntitySelection**

**.orderBy( pathObjects : Collection ) : 4D.EntitySelection** returns a new ordered entity selection containing all entities of the entity selection in the order specified by *pathString* or *pathObjects* criteria

**.orderByFormula( formulaString : Text { ; sortOrder : Integer } { ; settings : Object } ) : 4D.EntitySelection**

**.orderByFormula( formulaObj : Object { ; sortOrder : Integer } { ; settings : Object } ) : 4D.EntitySelection** returns a new, ordered entity selection

**.query( queryString : Text { ; ...value : any } { ; querySettings : Object } ) : 4D.EntitySelection**

**.query( formula : Object { ; querySettings : Object } ) : 4D.EntitySelection** searches for entities that meet the search criteria specified in *queryString* or *formula* and (optionally) *value*(s) among all the entities in the entity selection

**.queryPath : Text** contains a detailed description of the query as it was actually performed by 4D

**.queryPlan : Text** contains a detailed description of the query just before it is executed (i.e., the planned query)

**.refresh()** immediately "invalidates" the entity selection data in the local ORDA cache

**.selected( selectedEntities : 4D.EntitySelection ) : Object** returns an object describing the position(s) of *selectedEntities* in the original entity selection

**.slice( startFrom : Integer { ; end : Integer } ) : 4D.EntitySelection** returns a portion of an entity selection into a new entity selection

**.sum( attributePath : Text ) : Real** returns the sum for all *attributePath* values in the entity selection

**.toCollection( { options : Integer { ; begin : Integer { ; howMany : Integer } } } : Collection**

**.toCollection( filterString : Text { ; options : Integer { ; begin : Integer { ; howMany : Integer } } } ) : Collection**

**.toCollection( filterCol : Collection { ; options : Integer { ; begin : Integer { ; howMany : Integer } } } ) : Collection** creates and returns a collection where each element is an object containing a set of properties and values

## Create entity selection

**Create entity selection ( dsTable : Table { ; settings : Object } ) : 4D.EntitySelection**

| Parameter | Type                 | Description  |
|-----------|----------------------|--|
| dsTable   | Table                | -> Table in the 4D database whose current selection will be used to build the entity selection |
| settings  | Object               | -> Build option: context   |
| Result    | 4D.EntitySelection<- | Entity selection matching the dataclass related to the given table                             |

## Description

The `Create entity selection` command builds and returns a new, [alterable](#) entity selection related to the dataclass matching the given *dsTable*, according to the current selection of this table.

If the current selection is sorted, an [ordered](#) entity selection is created (the order of the current selection is kept). If the current selection is unsorted, an unordered entity selection is created.

If the *dsTable* is not exposed in [ds](#), an error is returned. This command cannot be used with a Remote datastore.

In the optional *settings* parameter, you can pass an object containing the following property:

| Property | Type | Description   |
|----------|------|---|
| context  | Text | Label for the <a href="#">optimization context</a> applied to the entity selection. |

## Example

```
var $employees : cs.EmployeeSelection
ALL RECORDS([Employee])
$employees:=Create entity selection([Employee])
// The $employees entity selection now contains a set of reference
// on all entities related to the Employee dataclass
```

## See also

[dataClass.newSelection\(\)](#)

# USE ENTITY SELECTION

## USE ENTITY SELECTION (*entitySelection*)

| Parameter       | Type            | Description            |
|-----------------|-----------------|------------------------|
| entitySelection | EntitySelection | -> An entity selection |

## Description

The `USE ENTITY SELECTION` command updates the current selection of the table matching the dataclass of the *entitySelection* parameter, according to the content of the entity selection.

This command cannot be used with a [Remote datastore](#).

After a call to `USE ENTITY SELECTION`, the first record of the updated current selection (if not empty) becomes the current record, but it is not loaded in memory. If you need to use the values of the fields in the current

record, use the `LOAD RECORD` command after the `USE ENTITY SELECTION` command.

## Example

```
var $entitySel : Object  
  
$entitySel:=ds.Employee.query("lastName = :1";"M@") // $entitySel is  
related to the Employee dataclass  
REDUCE SELECTION([Employee];0)  
USE ENTITY SELECTION($entitySel) //The current selection of the  
Employee table is updated
```

## [*index*]

- History

**[*index*]** : 4D.Entity

### Description

The `EntitySelection[index]` notation allows you to access entities within the entity selection using the standard collection syntax: pass the position of the entity you want to get in the *index* parameter.

Note that the corresponding entity is reloaded from the datastore.

*index* can be any number between 0 and `.length-1`.

- If *index* is out of range, an error is returned.
- If *index* corresponds to a dropped entity, a Null value is returned.

### ⚠ CAUTION

`EntitySelection[index]` is a non assignable expression, which means that it cannot be used as an editable entity reference with methods like `.lock()` or `.save()`. To work with the corresponding entity, you need to assign the returned expression to an assignable expression, such as a variable. Examples:

```
$sel:=ds.Employee.all() //create the entity selection  
//invalid statements:  
$result:=$sel[0].lock() //will NOT work  
$sel[0].lastName:="Smith" //will NOT work  
$result:=$sel[0].save() //will NOT work  
//valid code:  
$entity:=$sel[0] //OK  
$entity.lastName:="Smith" //OK  
$entity.save() //OK
```

## Example

```
var $employees : cs.EmployeeSelection  
var $employee : cs.EmployeeEntity  
$employees:=ds.Employee.query("lastName = :1";"H@")  
$employee:=$employees[2] // The 3rd entity of the $employees entity  
selection is reloaded from the database
```

## **.attributeName**

- History

**.attributeName** : Collection

**.attributeName** : 4D.EntitySelection

## Description

Any dataclass attribute can be used as a property of an entity selection to return a "projection" of values for the attribute in the entity selection. Projected values can be a collection or a new entity selection, depending on the [kind](#) (`storage` or `relation`) of the attribute.

- If `attributeName` kind is `storage`: `.attributeName` returns a collection of values of the same type as `attributeName`.
- If `attributeName` kind is `relatedEntity`: `.attributeName` returns a new entity selection of related values of the same type as `attributeName`. Duplications are removed (an unordered entity selection is returned).
- If `attributeName` kind is `relatedEntities`: `.attributeName` returns a new entity selection of related values of the same type as `attributeName`. Duplications are removed (an unordered entity selection is returned).

When a relation attribute is used as a property of an entity selection, the result is always another entity selection, even if only one entity is returned. In this case, if no entities are returned, the result is an empty entity selection.

If the attribute does not exist in the entity selection, an error is returned.

## Example 1

Projection of storage values:

```
var $firstNames : Collection
$entitySelection:=ds.Employee.all()
$firstNames:=$entitySelection.firstName // firstName type is string
```

The resulting collection is a collection of strings, for example:

```
[  
    "Joanna",  
    "Alexandra",  
    "Rick"  
]
```

## Example 2

Projection of related entity:

```
var $es; $entitySelection : cs.EmployeeSelection
$entitySelection:=ds.Employee.all()
$es:=$entitySelection.employer // employer is related to a Company
dataClass
```

The resulting object is an entity selection of Company with duplications removed (if any).

## Example 3

Projection of related entities:

```
var $es : cs.EmployeeSelection
$es:=ds.Employee.all().directReports // directReports is related to
Employee dataclass
```

The resulting object is an entity selection of Employee with duplications removed (if any).

## .add()

- History

`.add( entity : 4D.Entity ) : 4D.EntitySelection`

`.add( entitySelection : 4D.EntitySelection ) : 4D.EntitySelection`

| Parameter       | Type               | Description   |
|-----------------|--------------------|---|
| entity          | 4D.Entity          | - Entity to be added to the entity selection<br>>                                   |
| entitySelection | 4D.EntitySelection | - Entity selection to be added to the original entity<br>> selection                |
| Result          | 4D.EntitySelection | - Entity selection including the added <i>entity</i> or<br>> <i>entitySelection</i> |

### Description

The `.add()` function adds the specified *entity* or *entitySelection* to the original entity selection and returns the modified entity selection.

This function modifies the original entity selection.

#### ⚠ WARNING

The entity selection must be *alterable*, i.e. it has been created for example by `.newSelection()` or `Create entity selection`, otherwise `.add()` will return an error. Shareable entity selections do not accept the addition of entities. For more information, please refer to the [Shareable or alterable entity selections](#) section.

### Adding an entity

- If the entity selection is ordered, *entity* is added at the end of the selection. If a reference to the same entity already belongs to the entity selection, it is duplicated and a new reference is added.
- If the entity selection is unordered, *entity* is added anywhere in the selection, with no specific order.

### Adding an entity selection

- If the entity selection is ordered, its order is kept and *entitySelection* is added at the end of the selection. If references to the same entities of *entitySelection* already belong to the entity selection, they are duplicated and new references are added.
- If the entity selection is unordered, it becomes ordered.

For more information, please refer to the [Ordered or unordered entity selection](#) section.

The modified entity selection is returned by the function, so that function calls can be chained.

An error occurs if *entity* and the entity selection are not related to the same Dataclass. If *entity* is Null, no error is raised.

### Example 1

```

var $employees : cs.EmployeeSelection
var $employee : cs.EmployeeEntity
$employees:=ds.Employee.newSelection()
$employee:=ds.Employee.new()
$employee.lastName:="Smith"
$employee.save()
$employees.add($employee) //The $employee entity is added to the
$employees entity selection

```

## Example 2

Calls to the function can be chained:

```

var $sel : cs.ProductSelection
var $p1;$p2;$p3 : cs.ProductEntity

$p1:=ds.Product.get(10)
$p2:=ds.Product.get(11)
$p3:=ds.Product.get(12)
$sel:=ds.Product.newSelection()
$sel:=$sel.add($p1).add($p2).add($p3)

```

## Example 3

In a user interface, we have two lists. The user selects items from the list1 to add them to the list2.

```
$sellist2:=$sellist2.add($sellist1)
```

### .and()

- History

```

.and( entity : 4D.Entity ) : 4D.EntitySelection
.and( entitySelection : 4D.EntitySelection ) : 4D.EntitySelection

```

| Parameter       | Type               | Description  |
|-----------------|--------------------|--|
| entity          | 4D.Entity          | -> Entity to intersect with  |
| entitySelection | 4D.EntitySelection | -> Entity selection to intersect with  |
| Result          | 4D.EntitySelection | <- New entity selection with the result of intersection<br>with logical AND operator |

### Description

The `.and()` function combines the entity selection with an *entity* or *entitySelection* parameter using the logical AND operator; it returns a new, unordered entity selection that contains only the entities that are referenced in both the entity selection and the parameter.

- If you pass *entity* as parameter, you combine this entity with the entity selection. If the entity belongs to the entity selection, a new entity selection containing only the entity is returned. Otherwise, an empty entity selection is returned.
- If you pass *entitySelection* as parameter, you combine both entity selections. A new entity selection that contains only the entities that are referenced in both selections is returned. If there is no intersecting entity, an empty entity selection is returned.

You can compare [ordered and/or unordered entity selections](#). The resulting selection is always unordered.

If the original entity selection or the *entitySelection* parameter is empty, or if the *entity* is Null, an empty entity selection is returned.

If the original entity selection and the parameter are not related to the same dataclass, an error is raised.

## Example 1

```
var $employees; $result : cs.EmployeeSelection
var $employee : cs.EmployeeEntity
$employees:=ds.Employee.query("lastName = :1";"H@")
  //The $employees entity selection contains the entity
  //with primary key 710 and other entities
  //for ex. "Colin Hetrick" / "Grady Harness" / "Sherlock Holmes"
(primary key 710)
$employee:=ds.Employee.get(710) // Returns "Sherlock Holmes"

$result:=$employees.and($employee) // $result is an entity selection
containing
  //only the entity with primary key 710 ("Sherlock Holmes")
```

## Example 2

We want to have a selection of employees named "Jones" who live in New York:

```
var $sel1; $sel2; $sel3 : cs.EmployeeSelection
$sel1:=ds.Employee.query("name =:1";"Jones")
$sel2:=ds.Employee.query("city=:1";"New York")
$sel3:=$sel1.and($sel2)
```

## .at()

- History

**.at( index : Integer ) : 4D.Entity**

| Parameter | Type      | Description                 |
|-----------|-----------|-----------------------------|
| index     | Integer   | ->Index of entity to return |
| Result    | 4D.Entity | <-The entity at that index  |

## Description

The `.at()` function returns the entity at position *index*, allowing for positive and negative integer.

If *index* is negative (from -1 to -n with n : length of the entity selection), the returned entity will be based on the reverse order of the entity selection.

The function returns Null if *index* is beyond entity selection limits.

## Example

```
var $employees : cs.EmployeeSelection
var $emp1; $emp2 : cs.EmployeeEntity
$employees:=ds.Employee.query("lastName = :1";"H@")
$emp1:=$employees.at(2) //3rd entity of the $employees entity
selection
$emp2:=$employees.at(-3) //starting from the end, 3rd entity
/of the $employees entity selection
```

## .average()

- History

**.average( attributePath : Text ) : Real**

| Parameter     | Type | Description   |
|---------------|------|---|
| attributePath | Text | -> Attribute path to be used for calculation  |
| Result        | Real | <- Arithmetic mean (average) of entity attribute values (Undefined if empty entity selection) |

### Description

The `.average()` function returns the arithmetic mean (average) of all the non-null values of *attributePath* in the entity selection.

Pass in the *attributePath* parameter the attribute path to evaluate.

Only numerical values are taken into account for the calculation. Note however that, if the *attributePath* of the entity selection contains mixed value types, `.average()` takes all scalar elements into account to calculate the average value.

Date values are converted to numerical values (seconds) and used to calculate the average.

`.average()` returns **undefined** if the entity selection is empty or *attributePath* does not contain numerical values.

An error is returned if:

- *attributePath* is a related attribute,
- *attributePath* designates an attribute that does not exist in the entity selection dataclass.

### Example

We want to obtain a list of employees whose salary is higher than the average salary:

```
var $averageSalary : Real
var $moreThanAv : cs.EmployeeSelection
$averageSalary:=ds.Employee.all().average("salary")
$moreThanAv:=ds.Employee.query("salary > :1";$averageSalary)
```

## .contains()

- History

**.contains( entity : 4D.Entity ) : Boolean**

| Parameter | Type                          | Description   |
|-----------|-------------------------------|---|
| entity    | 4D.Entity->Entity to evaluate |   |
| Result    | Boolean                       | <- True if the entity belongs to the entity selection, else False |

### Description

The `.contains()` function returns true if entity reference belongs to the entity selection, and false otherwise.

In *entity*, specify the entity to search for in the entity selection. If entity is Null, the function will return false.

If *entity* and the entity selection do not belong to the same dataclass, an error is raised.

## Example

```
var $employees : cs.EmployeeSelection
var $employee : cs.EmployeeEntity

$employees:=ds.Employee.query("lastName=:1"; "H@")
$employee:=ds.Employee.get(610)

If($employees.contains($employee))
    ALERT("The entity with primary key 610 has a last name beginning
with H")
Else
    ALERT("The entity with primary key 610 does not have a last name
beginning with H")
End if
```

## .count()

- History

**.count( *attributePath* : Text ) : Real**

| <b>Parameter Type</b> | <b>Description</b>  |
|-----------------------|---|
| attributePath Text    | -> Path of the attribute to be used for calculation                       |
| Result      Real      | <- Number of non null <i>attributePath</i> values in the entity selection |

### Description

The `.count()` function returns the number of entities in the entity selection with a non-null value in *attributePath*.

Only scalar values are taken into account. Object or collection type values are considered as null values.

An error is returned if:

- *attributePath* is a related attribute,
- *attributePath* is not found in the entity selection dataclass.

## Example

We want to find out the total number of employees for a company without counting any whose job title has not been specified:

```
var $sel : cs.EmployeeSelection
var $count : Real

$sel:=ds.Employee.query("employer = :1"; "Acme, Inc")
$count:=$sel.count("jobtitle")
```

## .copy()

- History

**.copy( { option : Integer } ) : 4D.EntitySelection**

| Parameter | Type               | Description   |
|-----------|--------------------|---|
| option    | Integer            | -> <code>ck shared</code> : return a shareable entity selection |
| Result    | 4D.EntitySelection | <- Copy of the entity selection                                 |

## Description

The `.copy()` function returns a copy of the original entity selection.

This function does not modify the original entity selection.

By default, if the *option* parameter is omitted, the function returns a new, alterable entity selection (even if the function is applied to a shareable entity selection). Pass the `ck shared` constant in the *option* parameter if you want to create a shareable entity selection.

For information on the shareable property of entity selections, please refer to the [Shareable or alterable entity selections](#) section.

## Example

You create a new, empty entity selection of products when the form is loaded:

```
Case of
  :(Form event code=On Load)
    Form.products:=ds.Products.newSelection()
End case
```

Then this entity selection is updated with products and you want to share the products between several processes. You copy the Form.products entity selection as a shareable one:

```
...
// The Form.products entity selection is updated
Form.products.add(Form.selectedProduct)

Use(Storage)
If(Storage.products=NULL)
  Storage.products:=New shared object()
End if

Use(Storage.products)
  Storage.products:=Form.products.copy(ck shared)
End use
End use
```

## .distinct()

- History

**.distinct( attributePath : Text { ; options : Integer } ) : Collection**

| Parameter     | Type       | Description   |
|---------------|------------|---|
| attributePath | Text       | -> Path of attribute whose distinct values you want to get    |
| options       | Integer    | -> <code>dk diacritical</code> , <code>dk count values</code> |
| Result        | Collection | <- Collection with only distinct values                       |

## Description

The `.distinct()` function returns a collection containing only distinct (different) values from the *attributePath* in the entity selection.

The returned collection is automatically sorted. **Null** values are not returned.

In the *attributePath* parameter, pass the entity attribute whose distinct values you want to get. Only scalar values (text, number, boolean, or date) can be handled. If the *attributePath* leads to an object property that contains values of different types, they are first grouped by type and sorted afterwards. Types are returned in the following order:

1. booleans
2. strings
3. numbers
4. dates

You can use the `[]` notation to designate a collection when *attributePath* is a path within an object (see examples).

In the *options* parameter, you can pass one or a combination of the following constants:

| Constant                     | Value | Comment  |
|------------------------------|-------|--|
| <code>dk</code>              | 8     | Evaluation is case sensitive and differentiates accented characters.   |
| <code>diacritical</code>     |       | By default if omitted, a non-diacritical evaluation is performed   |
| <code>dk count values</code> | 32    | Return the count of entities for every distinct value. When this option is passed, <code>.distinct()</code> returns a collection of objects containing a pair of <code>{"value": *value*, "count": *count*}</code> properties. |

### NOTE

The `dk count values` option is only available with storage attributes of type boolean, string, number, and date.

An error is returned if:

- *attributePath* is a related attribute,
- *attributePath* is not found in the entity selection dataclass.

## Examples

You want to get a collection containing a single element per country name:

```
var $countries : Collection
$countries:=ds.Employee.all().distinct("address.country")
//$countries[0]={"Argentina"}
//$countries[1]={"Australia"}
//$countries[2]={"Belgium"}
///...
```

`nicknames` is a collection and `extra` is an object attribute:

```
$values:=ds.Employee.all().distinct("extra.nicknames[].first")
```

You want to get the number of different job names in the company:

```

var $jobs : Collection
$jobs:=ds.Employee.all().distinct("jobName";dk count values)
//$jobs[0]={ "value": "Developer", "count": 17}
//$jobs[1]={ "value": "Office manager", "count": 5}
//$jobs[2]={ "value": "Accountant", "count": 2}
//...

```

## .distinctPaths()

- History

**.distinctPaths( *attribute* : Text ) : Collection**

| Parameter | Type       | Description  |
|-----------|------------|--|
| attribute | Text       | -> Object attribute name whose paths you want to get |
| Result    | Collection | <- New collection with distinct paths                |

### Description

The `.distinctPaths()` function returns a collection of distinct paths found in the indexed object *attribute* for the entity selection.

If *attribute* is not an indexed object attribute, an error is generated.

After the call, the size of the returned collection is equal to the number of distinct paths found in *attribute* for the entity selection. Paths are returned as strings including nested attributes and collections, for example "info.address.number" or "children[].birthdate". Entities with a null value in the *attribute* are not taken into account.

### Example

You want to get all paths stored in a *fullData* object attribute:

```

var $paths : Collection
$paths:=ds.Employee.all().distinctPaths("fullData")
//$paths[0]="age"
//$paths[1]="Children"
//$paths[2]="Children[].age"
//$paths[3]="Children[].name"
//$paths[4]="Children.length"
///...

```



NOTE

*length* is automatically added as path for nested collection properties.

## .drop()

- History

**.drop( { mode : Integer } ) : 4D.EntitySelection**

| Parameter | Type               | Description  |
|-----------|--------------------|--|
| mode      | Integer            | -> <code>dk stop dropping on first error</code> : stops method execution on first non-droppable entity |
| Result    | 4D.EntitySelection | <- Empty entity selection if successful, else entity selection containing non-droppable entity(ies)    |

## Description

The `.drop()` function removes the entities belonging to the entity selection from the table related to its dataclass within the datastore. The entity selection remains in memory.

Removing entities is permanent and cannot be undone. It is recommended to call this action in a transaction in order to have a rollback option.

If a locked entity is encountered during the execution of `.drop()`, it is not removed. By default, the method processes all entities of the entity selection and returns non-droppable entities in the entity selection. If you want the method to stop execution at the first encountered non-droppable entity, pass the `dk stop dropping on first error` constant in the *mode* parameter.

## Example

Example without the `dk stop dropping on first error` option:

```
var $employees; $notDropped : cs.EmployeeSelection
$employees:=ds.Employee.query("firstName=:1";"S@")
$notDropped:=$employees.drop() // $notDropped is an entity selection
containing all the not dropped entities
If($notDropped.length=0) //The delete action is successful, all the
entities have been deleted
    ALERT("You have dropped "+String($employees.length)+" employees")
//The dropped entity selection remains in memory
Else
    ALERT("Problem during drop, try later")
End if
```

Example with the `dk stop dropping on first error` option:

```
var $employees; $notDropped : cs.EmployeeSelection
$employees:=ds.Employee.query("firstName=:1";"S@")
$notDropped:=$employees.drop(dk stop dropping on first error)
//$notDropped is an entity selection containing the first not dropped
entity
If($notDropped.length=0) //The delete action is successful, all the
entities have been deleted
    ALERT("You have dropped "+String($employees.length)+" employees")
//The dropped entity selection remains in memory
Else
    ALERT("Problem during drop, try later")
End if
```

## .extract()

- History

**.extract( *attributePath* : Text { ; *option* : Integer } ) : Collection**  
**.extract( *attributePath* { ; *targetPath* } { ; ...*attributePathN* : Text ; *targetPathN* : Text } ) : Collection**

| Parameter     | Type       | Description  |
|---------------|------------|--|
| attributePath | Text       | -> Attribute path whose values must be extracted to the new collection                                 |
| targetPath    | Text       | -> Target attribute path or attribute name   |
| option        | Integer    | -> <code>ck keep null</code> : include null attributes in the returned collection (ignored by default) |
| Result        | Collection | <- Collection containing extracted values  |

## Description

The `.extract()` function returns a collection containing *attributePath* values extracted from the entity selection.

*attributePath* can refer to:

- a scalar dataclass attribute,
- related entity,
- related entities.

If *attributePath* is invalid, an empty collection is returned.

This function accepts two syntaxes.

### **.extract( attributePath : Text { ; option : Integer } ) : Collection**

With this syntax, `.extract()` populates the returned collection with the *attributePath* values of the entity selection.

By default, entities for which *attributePath* is *null* or undefined are ignored in the resulting collection. You can pass the `ck keep null` constant in the *option* parameter to include these values as **null** elements in the returned collection.

- Dataclass attributes with `.kind` = "relatedEntity" are extracted as a collection of entities (duplications are kept).
- Dataclass attributes with `.kind` = "relatedEntities" are extracted as a collection of entity selections.

### **.extract( attributePath ; targetPath { ; ...attributePathN ; ... targetPathN} ) : Collection**

With this syntax, `.extract()` populates the returned collection with the *attributePath* properties. Each element of the returned collection is an object with *targetPath* properties filled with the corresponding *attributePath* properties. Null values are kept (*option* parameter is ignored with this syntax).

If several *attributePath* are given, a *targetPath* must be given for each. Only valid pairs [*attributePath*, *targetPath*] are extracted.

- Dataclass attributes with `.kind` = "relatedEntity" are extracted as an entity.
- Dataclass attributes with `.kind` = "relatedEntities" are extracted as an entity selection.

Entities of a collection of entities accessed by [ ] are not reloaded from the database.

## Example

Given the following table and relation:

```

var $firstnames; $addresses; $mailing; $teachers : Collection
//
//
// $firstnames is a collection of Strings

$firstnames:=ds.Teachers.all().extract("firstname")
//
// $addresses is a collection of entities related to dataclass Address
// Null values for address are extracted
$addresses:=ds.Teachers.all().extract("address";ck keep null)
//
//
// $mailing is a collection of objects with properties "who" and "to"
// "who" property content is String type
// "to" property content is entity type (Address dataclass)
$mailing:=ds.Teachers.all().extract("lastname";"who";"address";"to")
//
//
// $mailing is a collection of objects with properties "who" and "city"
// "who" property content is String type
// "city" property content is String type
$mailing:=ds.Teachers.all().extract("lastname";"who";"address.city";"c
)
//
// $teachers is a collection of objects with properties "where" and "w
// "where" property content is String
// "who" property content is an entity selection (Teachers dataclass)
$teachers:=ds.Address.all().extract("city";"where";"teachers";"who")
//
// $teachers is a collection of entity selections
$teachers:=ds.Address.all().extract("teachers")

```

## .first()

- History

**.first() : 4D.Entity**

| Parameter | Type         | Description  |
|-----------|--------------|--|
| Result    | 4D.Entity <- | Reference to the first entity of the entity selection (Null if selection is empty) |

### Description

The `.first()` function returns a reference to the entity in the first position of the entity selection.

The result of this function is similar to:

```
$entity:=$entitySel[0]
```

There is, however, a difference between both statements when the selection is empty:

```

var $entitySel : cs.EmpSelection
var $entity : cs.EmpEntity
$entitySel:=ds.Emp.query("lastName = :1";"Nonexistentname") //no
matching entity
//entity selection is then empty
$entity:=$entitySel.first() //returns Null
$entity:=$entitySel[0] //generates an error

```

## Example

```

var $entitySelection : cs.EmpSelection
var $entity : cs.EmpEntity
$entitySelection:=ds.Emp.query("salary > :1";100000)
If($entitySelection.length#0)
    $entity:=$entitySelection.first()
End if

```

## .getDataClass()

- History

### .getDataClass() : 4D.DataClass

| Parameter | Type            | Description  |
|-----------|-----------------|--|
| Result    | 4D.DataClass <- | Dataclass object to which the entity selection belongs |

## Description

The `.getDataClass()` function returns the dataclass of the entity selection.

This function is mainly useful in the context of generic code.

## Example

The following generic code duplicates all entities of the entity selection:

```

//duplicate_entities method
//duplicate_entities($entity_selection)

#DECLARE ( $entitySelection : 4D.EntitySelection )
var $dataClass : 4D.DataClass
var $entity; $duplicate : 4D.Entity
var $status : Object
$dataClass:=$entitySelection.getDataClass()
For each($entity;$entitySelection)
    $duplicate:=$dataClass.new()
    $duplicate.fromObject($entity.toObject())
    $duplicate[$dataClass.getInfo().primaryKey]:=Null //reset the
primary key
    $status:=$duplicate.save()
End for each

```

## .getRemoteContextAttributes()

- History

### .getRemoteContextAttributes() : Text

| Parameter Type      | Description   |
|---------------------|---|
| result      Text <- | Context attributes linked to the entity selection, separated by a comma |

**Advanced mode:** This function is intended for developers who need to customize ORDA default features for specific configurations. In most cases, you will not need to use it.

## Description

The `.getRemoteContextAttributes()` function returns information about the optimization context used by the entity selection.

If there is no [optimization context](#) for the entity selection, the function returns an empty Text.

## Example

```
var $ds : 4D.DataStoreImplementation
var $persons : cs.PersonsSelection
var $p : cs.PersonsEntity

var $info : Text
var $text : Text

$ds:=Open datastore(New object("hostname"; "www.myserver.com"); "myDS")

$persons:=$ds.Persons.all()
$text:=""
For each ($p; $persons)
    $text:=$p.firstname+" lives in "+$p.address.city+" / "
End for each

$info:=$persons.getRemoteContextAttributes()
//$info = "firstname,address,address.city"
```

## See also

[Entity.getRemoteContextAttributes\(\)](#)  
[.clearAllRemoteContexts\(\)](#)  
[.getRemoteContextInfo\(\)](#)  
[.getAllRemoteContexts\(\)](#)  
[.setRemoteContextInfo\(\)](#)

## .isAlterable()

- History

**.isAlterable()** : Boolean

| Parameter Type         | Description  |
|------------------------|--|
| Result      Boolean <- | True if the entity selection is alterable, False otherwise |

## Description

The `.isAlterable()` function returns True if the entity selection is alterable, and False if the entity selection is not alterable.

For more information, please refer to [Shareable or alterable entity selections](#).

## Example

You are about to display `Form.products` in a [list box](#) to allow the user to add new products. You want to make sure it is alterable so that the user can add new products without error:

```
If (Not (Form.products.isAlterable()))
    Form.products:=Form.products.copy()
End if
...
Form.products.add (Form.product)
```

## .isOrdered()

- History

**.isOrdered()** : Boolean

| Parameter | Type       | Description  |
|-----------|------------|--|
| Result    | Boolean <- | True if the entity selection is ordered, False otherwise |

### Description

The `.isOrdered()` function returns True if the entity selection is ordered, and False if it is unordered.

This function always returns True when the entity selection comes from a remote datastore.

For more information, please refer to [Ordered or unordered entity selection](#).

## Example

```
var $employees : cs.EmployeeSelection
var $employee : cs.EmployeeEntity
var $isOrdered : Boolean
$employees:=ds.Employee.newSelection(dk keep ordered)
$employee:=ds.Employee.get(714) // Gets the entity with primary key
714

//In an ordered entity selection, we can add the same entity several
times (duplications are kept)
$employees.add($employee)
$employees.add($employee)
$employees.add($employee)

$isOrdered:=$employees.isOrdered()
If ($isOrdered)
    ALERT("The entity selection is ordered and contains
"+String($employees.length)+" employees")
End if
```

## .last()

- History

**.last()** : 4D.Entity

| Parameter | Type         | Description   |
|-----------|--------------|---|
| Result    | 4D.Entity <- | Reference to the last entity of the entity selection (Null if empty entity selection) |

## Description

The `.last()` function returns a reference to the entity in last position of the entity selection.

The result of this function is similar to:

```
$entity:=$entitySel[length-1]
```

If the entity selection is empty, the function returns Null.

## Example

```
var $entitySelection : cs.EmpSelection
var $entity : cs.EmpEntity
$entitySelection:=ds.Emp.query("salary < :1";50000)
If($entitySelection.length#0)
    $entity:=$entitySelection.last()
End if
```

## .length

- History

**.length** : Integer

## Description

The `.length` property returns the number of entities in the entity selection. If the entity selection is empty, it returns 0.

Entity selections always have a `.length` property.

To know the total number of entities in a dataclass, it is recommended to use the [getCount\(\)](#) function which is more optimized than the `ds.myClass.all().length` expression.

## Example

```
var $vSize : Integer
$vSize:=ds.Employee.query("gender = :1;"male").length
ALERT(String(vSize)+" male employees found.")
```

## .max()

- History

**.max( attributePath : Text ) : any**

| Parameter     | Type    | Description                                      |
|---------------|---------|--|
| attributePath | Text -> | Path of the attribute to be used for calculation |
| Result        | any     | <-Highest value of attribute                     |

## Description

The `.max()` function returns the highest (or maximum) value among all the values of *attributePath* in the entity selection. It actually returns the value of the last entity of the entity selection as it would be sorted in ascending order using the [.orderBy\(\)](#) function.

If you pass in *attributePath* a path to an object property containing different types of values, the `.max()` function will return the maximum value within the first scalar type in the default 4D type list order (see [.sort\(\)](#) description).

`.max()` returns **undefined** if the entity selection is empty or *attributePath* is not found in the object attribute.

An error is returned if:

- *attributePath* is a related attribute,
- *attributePath* designates an attribute that does not exist in the entity selection dataclass.

## Example

We want to find the highest salary among all the female employees:

```
var $sel : cs.EmpSelection
var $maxSalary : Real
$sel:=ds.Employee.query("gender = :1";"female")
$maxSalary:=$sel.max("salary")
```

## .min()

▪ History

**.min( *attributePath* : Text ) : any**

| Parameter Type        | Description                                      |
|-----------------------|--|
| attributePath Text -> | Path of the attribute to be used for calculation |
| Result any            | <- Lowest value of attribute                     |

## Description

The `.min()` function returns the lowest (or minimum) value among all the values of *attributePath* in the entity selection. It actually returns the first entity of the entity selection as it would be sorted in ascending order using the [.orderBy\(\)](#) function (excluding **null** values).

If you pass in *attributePath* a path to an object property containing different types of values, the `.min()` function will return the minimum value within the first scalar value type in the type list order (see [.sort\(\)](#) description).

`.min()` returns **undefined** if the entity selection is empty or *attributePath* is not found in the object attribute.

An error is returned if:

- *attributePath* is a related attribute,
- *attributePath* designates an attribute that does not exist in the entity selection dataclass.

## Example

In this example, we want to find the lowest salary among all the female employees:

```
var $sel : cs.EmpSelection
var $minSalary : Real
$sel:=ds.Employee.query("gender = :1;";"female")
$minSalary:=$sel.min("salary")
```

## .minus()

- History

**.minus( entity : 4D.Entity { ; keepOrder : Integer } ) : 4D.EntitySelection**  
**.minus( entitySelection : 4D.EntitySelection { ; keepOrder : Integer } ) : 4D.EntitySelection**

| Parameter       | Type                  | Description   |
|-----------------|-----------------------|---|
| entity          | 4D.Entity             | -> Entity to subtract   |
| entitySelection | 4D.EntitySelection    | -> Entity selection to subtract   |
| keepOrder       | Integer               | -> <code>dk keep ordered</code> (integer) to keep the initial order in the resulting entity selection |
| Result          | 4D.EntitySelection <- | New entity selection or a new reference on the existing entity selection                              |

## Description

The `.minus()` function excludes from the entity selection to which it is applied the *entity* or the entities of *entitySelection* and returns the resulting entity selection.

- If you pass *entity* as parameter, the function creates a new entity selection without *entity* (if *entity* belongs to the entity selection). If *entity* was not included in the original entity selection, a new reference to the entity selection is returned.
- If you pass *entitySelection* as parameter, the function returns an entity selection containing the entities belonging to the original entity selection without the entities belonging to *entitySelection*. You can compare [ordered and/or unordered entity selections](#).

By default, if you omit the *keepOrder* parameter, the resulting entity selection is unordered. If you want to keep the order of the original entity selection (for example if you want to reuse the entity selection in a user interface), pass the `dk keep ordered` constant in *keepOrder*. In this case, the result is an ordered entity selection and the order of the initial entity selection is kept.

### NOTE

If you pass `dk keep ordered` in *keepOrder* and the removed *entitySelection* contains entities duplicated in the original entity selection, all occurrences of the duplicates are removed.

If the original entity selection or both the original entity selection and the *entitySelection* parameter are empty, an empty entity selection is returned.

If *entitySelection* is empty or if *entity* is Null, a new reference to the original entity selection is returned.

If the original entity selection and the parameter are not related to the same dataclass, an error is raised.

## Example 1

```
var $employees; $result : cs.EmployeeSelection
var $employee : cs.EmployeeEntity

$employees:=ds.Employee.query("lastName = :1";"H@")
// The $employees entity selection contains the entity with primary
key 710 and other entities
// for ex. "Colin Hetrick", "Grady Harness", "Sherlock Holmes"
(primary key 710)

$employee:=ds.Employee.get(710) // Returns "Sherlock Holmes"

$result:=$employees.minus($employee) // $result contains "Colin
Hetrick", "Grady Harness"
```

## Example 2

We want to have a selection of female employees named "Jones" who live in New York :

```
var $sel1; $sel2; $sel3 : cs.EmployeeSelection
$sel1:=ds.Employee.query("name =:1";"Jones")
$sel2:=ds.Employee.query("city=:1";"New York")
$sel3:=$sel1.and($sel2).minus(ds.Employee.query("gender='male'"))
```

## Example 3

In a user interface, we have a list that displays items in a specific order. If the user selects items in the list to remove them, the order must be kept when refreshing the list:

```
$listsel:=$listsel.minus($selectedItems; dk keep ordered)
```

## .or()

- History

```
.or( entity : 4D.Entity ) : 4D.EntitySelection
.or( entitySelection : 4D.EntitySelection ) : 4D.EntitySelection
```

| Parameter       | Type                  | Description  |
|-----------------|-----------------------|--|
| entity          | 4D.Entity             | -> Entity to intersect with  |
| entitySelection | 4D.EntitySelection    | -> Entity selection to intersect with                                  |
| Result          | 4D.EntitySelection <- | New entity selection or new reference to the original entity selection |

## Description

The `.or()` function combines the entity selection with the *entity* or *entitySelection* parameter using the logical (not exclusive) OR operator; it returns a new, unordered entity selection that contains all the entities from the entity selection and the parameter.

- If you pass *entity* as parameter, you compare this entity with the entity selection. If the entity belongs to the entity selection, a new reference to the entity selection is returned. Otherwise, a new entity selection containing the original entity selection and the entity is returned.
- If you pass *entitySelection* as parameter, you compare entity selections. A new entity selection containing the entities belonging to the original entity selection or

*entitySelection* is returned (or is not exclusive, entities referenced in both selections are not duplicated in the resulting selection).

You can compare [ordered and/or unordered entity selections](#). The resulting selection is always unordered.

If the original entity selection and the *entitySelection* parameter are empty, an empty entity selection is returned. If the original entity selection is empty, a reference to *entitySelection* or an entity selection containing only *entity* is returned.

If *entitySelection* is empty or if *entity* is Null, a new reference to the original entity selection is returned.

If the original entity selection and the parameter are not related to the same dataclass, an error is raised.

## Example 1

```
var $employees1; $employees2; $result : cs.EmployeeSelection
$employees1:=ds.Employee.query("lastName = :1";"H@") //Returns "Colin
Hetrick", "Grady Harness"
$employees2:=ds.Employee.query("firstName = :1";"C@") //Returns "Colin
Hetrick", "Cath Kidston"
$result:=$employees1.or($employees2) // $result contains "Colin
Hetrick", "Grady Harness", "Cath Kidston"
```

## Example 2

```
var $employees; $result : cs.EmployeeSelection
var $employee : cs.EmployeeEntity
$employees:=ds.Employee.query("lastName = :1";"H@") // Returns "Colin
Hetrick", "Grady Harness", "Sherlock Holmes"
$employee:=ds.Employee.get(686) //the entity with primary key 686 does
not belong to the $employees entity selection
//It matches the employee "Mary Smith"

$result:=$employees.or($employee) // $result contains "Colin Hetrick",
"Grady Harness", "Sherlock Holmes", "Mary Smith"
```

## .orderBy()

- History

**.orderBy( pathString : Text ) : 4D.EntitySelection**  
**.orderBy( pathObjects : Collection ) : 4D.EntitySelection**

| Parameter   | Type               | Description  |
|-------------|--------------------|--|
| pathString  | Text               | -> Attribute path(s) and sorting instruction(s) for the entity selection |
| pathObjects | Collection         | -> Collection of criteria objects  |
| Result      | 4D.EntitySelection | <- New entity selection in the specified order                           |

## Description

The `.orderBy()` function returns a new ordered entity selection containing all entities of the entity selection in the order specified by *pathString* or *pathObjects* criteria.

- This method does not modify the original entity selection.
- For more information on ordered entity selections, please refer to the

## [Ordered or unordered entity selection](#) section.

You must use a criteria parameter to define how the entities must be sorted. Two different parameters are supported:

- *pathString* (Text) : This parameter contains a formula made of 1 to x attribute paths and (optionally) sort orders, separated by commas. The syntax is:

```
"attributePath1 {desc or asc}, attributePath2 {desc or asc},..."
```

The order in which the attributes are passed determines the sorting priority of the entities. By default, attributes are sorted in ascending order. You can set the sort order of a property in the criteria string, separated from the property path by a single space: pass "asc" to sort in ascending order or "desc" in descending order.

- *pathObjects* (collection): each element of the collection contains an object structured in the following way:

```
{
    "propertyPath": string,
    "descending": boolean
}
```

By default, attributes are sorted in ascending order ("descending" is false).

You can add as many objects in the criteria collection as necessary.

Null values are evaluated as less than other values.

If you pass an invalid attribute path in *pathString* or *pathObject*, the function returns an empty entity selection.

## Example

```
// order by formula
$sortedEntitySelection:=$entitySelection.orderBy("firstName asc,
salary desc")
$sortedEntitySelection:=$entitySelection.orderBy("firstName")

// order by collection with or without sort orders
$orderColl:=New collection
$orderColl.push(New
object("propertyPath";"firstName";"descending";False))
$orderColl.push(New
object("propertyPath";"salary";"descending";True))
$sortedEntitySelection:=$entitySelection.orderBy($orderColl)

$orderColl:=New collection
$orderColl.push(New object("propertyPath";"manager.lastName"))
$orderColl.push(New object("propertyPath";"salary"))
$sortedEntitySelection:=$entitySelection.orderBy($orderColl)
```

## .orderByFormula()

- History

```
.orderByFormula( formulaString : Text { ; sortOrder : Integer } { ; settings :
Object} ) : 4D.EntitySelection
.orderByFormula( formulaObj : Object { ; sortOrder : Integer } { ; settings : Object}
) : 4D.EntitySelection
```

| Parameter     | Type               | Description                                |
|---------------|--------------------|--|
| formulaString | Text               | -> Formula string                          |
| formulaObj    | Object             | -> Formula object                          |
| sortOrder     | Integer            | -> dk ascending (default) or dk descending |
| settings      | Object             | -> Parameter(s) for the formula            |
| Result        | 4D.EntitySelection | <- New ordered entity selection            |

## Description

The `.orderByFormula()` function returns a new, ordered entity selection containing all entities of the entity selection in the order defined through the *formulaString* or *formulaObj* and, optionally, *sortOrder* and *settings* parameters.

This function does not modify the original entity selection.

You can use either a *formulaString* or a *formulaObj* parameter:

- *formulaString*: you pass a 4D expression such as "Year of(this.birthDate)".
- *formulaObj*: pass a valid formula object created using the `Formula` or `Formula from string` command.

The *formulaString* or *formulaObj* is executed for each entity of the entity selection and its result is used to define the position of the entity in the returned entity selection. The result must be of a sortable type (boolean, date, number, text, time, null).

A null result is always the smallest value.

By default if you omit the *sortOrder* parameter, the resulting entity selection is sorted in ascending order. Optionally, you can pass one of the following values in the *sortOrder* parameter:

| Constant      | Value | Comment                        |
|---------------|-------|--------------------------------|
| dk ascending  | 0     | Ascending sort order (default) |
| dk descending | 1     | Descending sort order          |

Within the *formulaString* or *formulaObj*, the processed entity and thus its attributes are available through the `This` command (for example, `This.lastName`).

You can pass parameter(s) to the formula using the `args` property (object) of the `settings` parameter: the formula receives the `settings.args` object in \$1.

## Example 1

Sorting students using a formula provided as text:

```
var $es1; $es2 : cs.StudentsSelection
$es1:=ds.Students.query("nationality=:1";"French")
$es2:=$es1.orderByFormula("length(this.lastname)") //ascending by
default
$es2:=$es1.orderByFormula("length(this.lastname)";dk descending)
```

Same sort order but using a formula object:

```

var $es1; $es2 : cs.StudentsSelection
var $formula : Object
$es1:=ds.Students.query("nationality=:1";"French")
$formula:=Formula(Length(This.lastname))
$es2:=$es1.orderByFormula($formula) // ascending by default
$es2:=$es1.orderByFormula($formula;dk descending)

```

## Example 2

A formula is given as a formula object with parameters; `settings.args` object is received as \$1 in the ***computeAverage*** method.

In this example, the "marks" object field in the **Students** dataClass contains students' grades for each subject. A single formula object is used to compute a student's average grade with different coefficients for schoolA and schoolB.

```

var $es1; $es2 : cs.StudentsSelection
var $formula; $schoolA; $schoolB : Object
$es1:=ds.Students.query("nationality=:1";"French")
$formula:=Formula(computeAverage($1))

$schoolA:=New object() //settings object
$schoolA.args:=New object("english";1;"math";1;"history";1) // 
Coefficients to compute an average

//Order students according to school A criteria
$es2:=$es1.entitySelection.orderByFormula($formula;$schoolA)

$schoolB:=New object() //settings object
$schoolB.args:=New object("english";1;"math";2;"history";3) // 
Coefficients to compute an average

//Order students according to school B criteria
$es2:=$es1.entitySelection.orderByFormula($formula;dk
descending;$schoolB)
//
// computeAverage method
// -----
#DECLARE ($coefList : Object) -> $result : Integer
var $subject : Text
var $average; $sum : Integer

$average:=0
$sum:=0

For each($subject;$coefList)
    $sum:=$sum+$coefList[$subject]
End for each

For each($subject;This.marks)
    $average:=$average+(This.marks[$subject]*$coefList[$subject])
End for each

$result:=$average/$sum

```

## .query()

- History

**.query( queryString : Text { ; ...value : any } { ; querySettings : Object } ) :**  
**4D.EntitySelection**  
**.query( formula : Object { ; querySettings : Object } ) : 4D.EntitySelection**

| Parameter           | Type               | Description   |
|---------------------|--------------------|---|
| queryString         | Text               | -> Search criteria as string  |
| formula             | Object             | -> Search criteria as formula object  |
| value               | any                | -> Value(s) to use for indexed placeholder(s)   |
| querySettingsObject |                    | -> Query options: parameters, attributes, args, allowFormulas, context, queryPath, queryPlan<br>New entity selection made up of entities from |
| Result              | 4D.EntitySelection | <- entity selection meeting the search criteria specified in <i>queryString</i> or <i>formula</i>   |

## Description

The `.query()` function searches for entities that meet the search criteria specified in *queryString* or *formula* and (optionally) *value(s)* among all the entities in the entity selection, and returns a new object of type `EntitySelection` containing all the entities that are found. Lazy loading is applied.

This function does not modify the original entity selection.

If no matching entities are found, an empty `EntitySelection` is returned.

For detailed information on how to build a query using *queryString*, *value*, and *querySettings* parameters, please refer to the DataClass [.query\(\)](#) function description.

By default if you omit the **order by** statement in the *queryString*, the returned entity selection is [not ordered](#). Note however that, in Client/Server mode, it behaves like an ordered entity selection (entities are added at the end of the selection).

## Example 1

```
var $entitySelectionTemp : cs.EmployeeSelection
$entitySelectionTemp:=ds.Employee.query("lastName = :1";"M@")
Form.emps:=$entitySelectionTemp.query("manager.lastName = :1";"S@")
```

## Example 2

More examples of queries can be found in the DataClass [.query\(\)](#) page.

## See also

[.query\(\)](#) for dataclass

## .queryPath

- History

**.queryPath** : Text

## Description

The `.queryPath` property contains a detailed description of the query as it was actually performed by 4D. This property is available for `EntitySelection` objects generated through queries if the `"queryPath":true` property was passed in the *querySettings* parameter of the [.query\(\)](#) function.

For more information, refer to the **querySettings parameter** paragraph in the Dataclass [.query\(\)](#) page.

## .queryPlan

- History

**.queryPlan** : Text

### Description

The `.queryPlan` property contains a detailed description of the query just before it is executed (i.e., the planned query). This property is available for `EntitySelection` objects generated through queries if the `"queryPlan":true` property was passed in the `querySettings` parameter of the [.query\(\)](#) function.

For more information, refer to the **querySettings parameter** paragraph in the Dataclass [.query\(\)](#) page.

## .refresh()

- History

**.refresh()**

| Parameter | Type                            | Description |
|-----------|---------------------------------|-------------|
|           | Does not require any parameters |             |

### Description

This function only works with a remote datastore (client / server or `Open datastore` connection).

The `.refresh()` function immediately "invalidates" the entity selection data in the local ORDA cache so that the next time 4D requires the entity selection, it will be reloaded from the database.

By default, the local ORDA cache is invalidated after 30 seconds. In the context of client / server applications using both ORDA and the classic language, this method allows you to make sure a remote application will always work with the latest data.

### Example 1

In this example, classic and ORDA code modify the same data simultaneously:

```

//On a 4D remote

var $selection : cs.StudentsSelection
var $student : cs.StudentsEntity

$selection:=ds.Students.query("lastname=:1";"Collins")
//The first entity is loaded in the ORDA cache
$student:=$selection.first()

//Update with classic 4D, ORDA cache is not aware of if
QUERY([Students];[Students]lastname="Collins")
[Students]lastname:="Colin"
SAVE RECORD([Students])

//to get the latest version, the ORDA cache must be invalidated
$selection.refresh()
// Even if cache is not expired, the first entity is reloaded from
disk
$student:=$selection.first()

// $student.lastname contains "Colin"

```

## Example 2

A list box displays the Form.students entity selection and several clients work on it.

```

// Form method:
Case of
  :(Form event code=On Load)
    Form.students:=ds.Students.all()
End case
//
//
// On client #1, the user loads, updates, and saves the first entity
// On client #2, the user loads, updates, and saves the same entity
//
//
// On client #1:
Form.students.refresh() // Invalidates the ORDA cache for the
Form.students entity selection
// The list box content is refreshed from the database with update
made by client #2

```

## .selected()

- History

**.selected( selectedEntities : 4D.EntitySelection ) : Object**

| Parameter        | Type                  | Description  |
|------------------|-----------------------|--|
| selectedEntities | 4D.EntitySelection -> | Entity selection with entities for which to know<br>the rank in the entity selection |
| Result           | Object                | <- Range(s) of selected entities in entity selection                                 |

## Description

The `.selected()` function returns an object describing the position(s) of `selectedEntities` in the original entity selection.

This function does not modify the original entity selection.

Pass in the `selectedEntities` parameter an entity selection containing entities for which

you want to know the position in the original entity selection. `selectedEntities` must be an entity selection belonging to the same dataclass as the original entity selection, otherwise an error 1587 - "The entity selection comes from an incompatible dataclass" is raised.

## Result

The returned object contains the following properties:

| Property  | Type       | Description                     |
|---|------------|---------------------------------|
| <code>ranges</code>                               | Collection | Collection of range objects     |
| <code>ranges[]</code> . <code>startInteger</code> |            | First entity index in the range |
| <code>ranges[]</code> . <code>end</code>          | Integer    | Last entity index in the range  |

If a `ranges` property contains a single entity, `start = end`. Index starts at 0.

The function returns an empty collection in the `ranges` property if the original entity selection or the `selectedEntities` entity selection is empty.

## Example

```
var $invoices; $cashSel; $creditSel : cs.Invoices
var $result1; $result2 : Object

$invoices:=ds.Invoices.all()

$cashSelection:=ds.Invoices.query("payment = :1"; "Cash")
$creditSel:=ds.Invoices.query("payment IN :1"; New collection("Cash";
"Credit Card"))

$result1:=$invoices.selected($cashSelection)
$result2:=$invoices.selected($creditSel)

//$result1 = {ranges:[{start:0;end:0},{start:3;end:3},
{start:6;end:6}]}
//$result2 = {ranges:[{start:0;end:1},{start:3;end:4},
{start:6;end:7}]}
```

## .slice()

- History

**.slice( *startFrom* : Integer { ; *end* : Integer } ) : 4D.EntitySelection**

| Parameter              | Type               | Description  |
|------------------------|--------------------|--|
| <code>startFrom</code> | Integer            | -> Index to start the operation at (included)                        |
| <code>end</code>       | Integer            | -> End index (not included)  |
| <code>Result</code>    | 4D.EntitySelection | <- New entity selection containing sliced entities<br>(shallow copy) |

## Description

The `.slice()` function returns a portion of an entity selection into a new entity selection, selected from the `startFrom` index to the `end` index (`end` is not included) or to the last entity of the entity selection. This method returns a shallow copy of the entity selection (it uses the same entity references).

This function does not modify the original entity selection.

The returned entity selection contains the entities specified by *startFrom* and all subsequent entities up to, but not including, the entity specified by *end*. If only the *startFrom* parameter is specified, the returned entity selection contains all entities from *startFrom* to the last entity of the original entity selection.

- If *startFrom* < 0, it is recalculated as *startFrom*:=*startFrom*+*length* (it is considered as the offset from the end of the entity selection). If the calculated value < 0, *startFrom* is set to 0.
- If *startFrom* >= *length*, the function returns an empty entity selection.
- If *end* < 0, it is recalculated as *end*:=*end*+*length*.
- If *end* < *startFrom* (passed or calculated values), the method does nothing.

If the entity selection contains entities that were dropped in the meantime, they are also returned.

## Example 1

You want to get a selection of the first 9 entities of the entity selection:

```
var $sel; $sliced : cs.EmployeeSelection  
$sel:=ds.Employee.query("salary > :1";50000)  
$sliced:=$sel.slice(0;9) //
```

## Example 2

Assuming we have `ds.Employee.all().length = 10`

```
var $slice : cs.EmployeeSelection  
$slice:=ds.Employee.all().slice(-1;-2) //tries to return entities from  
index 9 to 8, but since 9 > 8, returns an empty entity selection
```

## .sum()

- History

**.sum( *attributePath* : Text ) : Real**

| Parameter Type                     | Description                                      |
|------------------------------------|--|
| <code>attributePath</code> Text -> | Path of the attribute to be used for calculation |
| Result                             | Real <-Sum of entity selection values            |

### Description

The `.sum()` function returns the sum for all *attributePath* values in the entity selection.

`.sum()` returns 0 if the entity selection is empty.

The sum can only be done on values of number type. If the *attributePath* is an object property, only numerical values are taken into account for the calculation (other value types are ignored). In this case, if *attributePath* leads to a property that does not exist in the object or does not contain any numeric values, `.sum()` returns 0.

An error is returned if:

- *attributePath* is not a numerical or an object attribute,
- *attributePath* is a related attribute,
- *attributePath* is not found in the entity selection dataclass.

## Example

```
var $sel : cs.EmployeeSelection
var $sum : Real

$sel:=ds.Employee.query("salary < :1";20000)
$sum:=$sel.sum("salary")
```

## .toCollection()

- History

**.toCollection( { options : Integer { ; begin : Integer { ; howMany : Integer } } } ) : Collection**  
**.toCollection( filterString : Text {; options : Integer { ; begin : Integer { ; howMany : Integer } } } ) : Collection**  
**.toCollection( filterCol : Collection {; options : Integer { ; begin : Integer { ; howMany : Integer } } } ) : Collection**

| Parameter    | Type       | Description   |
|--------------|------------|---|
| filterString | Text       | -> String with entity attribute path(s) to extract  |
| filterCol    | Collection | -> Collection of entity attribute path(s) to extract  |
| options      | Integer    | -> <small>dk with primary key</small> : adds the primary key<br><small>dk with stamp</small> : adds the stamp |
| begin        | Integer    | -> Designates the starting index  |
| howMany      | Integer    | -> Number of entities to extract  |
| Result       | Collection | <- Collection of objects containing attributes and values of entity selection                                 |

## Description

The `.toCollection()` function creates and returns a collection where each element is an object containing a set of properties and values corresponding to the attribute names and values for the entity selection.

If no filter parameter is passed or the first parameter contains an empty string or "\*", all the attributes are extracted. Attributes with [kind](#) property as "relatedEntity" are extracted with the simple form: an object with property `__KEY` (primary key). Attributes with kind property as "relatedEntities" are not extracted.

Or, you can designate the entity attributes to extract using a filter parameter. You can use one of these two filters:

- *filterString* --a string with property paths separated with commas:  
"propertyPath1, propertyPath2, ...".
- *filterCol* --a collection of strings containing property paths:  
["propertyPath1","propertyPath2",...]

If a filter is specified for an attribute of the `relatedEntity` kind:

- *propertyPath* = "relatedEntity" -> it is extracted with simple form
- *propertyPath* = "relatedEntity.\*" -> all the properties are extracted
- *propertyPath* = "relatedEntity.propertyName1, relatedEntity.propertyName2, ..." -> only those properties are extracted

If a filter is specified for an attribute of the `relatedEntities` kind:

- `propertyPath = "relatedEntities.*"` -> all the properties are extracted
- `propertyPath = "relatedEntities.propertyName1, relatedEntities.propertyName2, ..."` -> only those properties are extracted

In the `options` parameter, you can pass the `dk with primary key` and/or `dk with stamp` selector(s) to add the entity's primary keys and/or stamps in extracted objects.

## ⚠ WARNING

If you use another attribute than the primary key as the One attribute in a relation, the value of this attribute will be written in the "`__KEY`" property. Keep in mind that it is recommended to use the primary key as One attribute in your relations, especially when you use `.toCollection()` and `.fromCollection()` functions.

The `begin` parameter allows you to indicate the starting index of the entities to extract. You can pass any value between 0 and entity selection length-1.

The `howMany` parameter lets you specify the number of entities to extract, starting with the one specified in `begin`. Dropped entities are not returned but are taken into account according to `howMany`. For example, if `howMany= 3` and there is 1 dropped entity, only 2 entities are extracted.

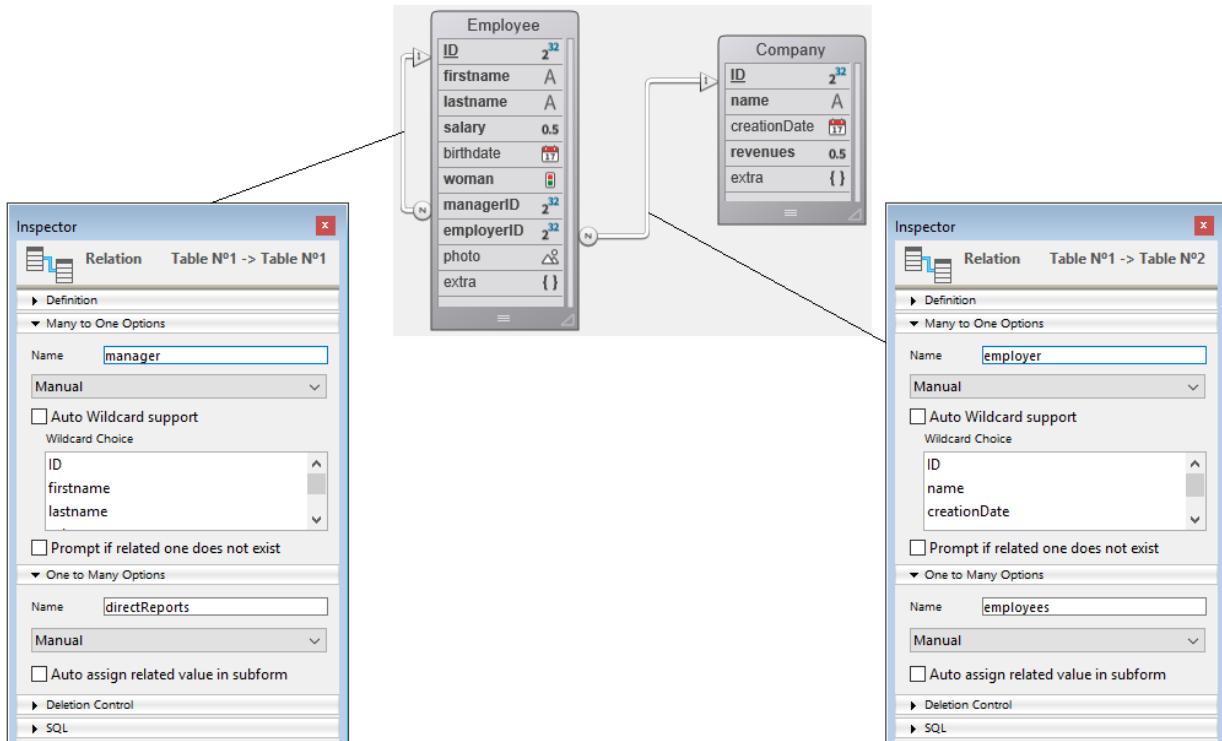
If `howMany > length` of the entity selection, the method returns `(length - begin)` objects.

An empty collection is returned if:

- the entity selection is empty, or
- `begin` is greater than the length of the entity selection.

## Example 1

The following structure will be used throughout all examples of this section:



Example without filter or options parameter:

```

var $employeesCollection : Collection
var $employees : cs.EmployeeSelection

$employeesCollection:=New collection
$employees:=ds.Employee.all()
$employeesCollection:=$employees.toCollection()

```

**Returns:**

```

[
  {
    "ID": 416,
    "firstName": "Gregg",
    "lastName": "Wahl",
    "salary": 79100,
    "birthDate": "1963-02-01T00:00:00.000Z",
    "woman": false,
    "managerID": 412,
    "employerID": 20,
    "photo": "[object Picture]",
    "extra": null,
    "employer": {
      "__KEY": 20
    },
    "manager": {
      "__KEY": 412
    }
  },
  {
    "ID": 417,
    "firstName": "Irma",
    "lastName": "Durham",
    "salary": 47000,
    "birthDate": "1992-06-16T00:00:00.000Z",
    "woman": true,
    "managerID": 412,
    "employerID": 20,
    "photo": "[object Picture]",
    "extra": null,
    "employer": {
      "__KEY": 20
    },
    "manager": {
      "__KEY": 412
    }
  }
]

```

## Example 2

**Example with options:**

```

var $employeesCollection : Collection
var $employees : cs.EmployeeSelection

$employeesCollection:=New collection
$employees:=ds.Employee.all()
$employeesCollection:=$employees.toCollection("");dk with primary key+dk
with stamp)

```

**Returns:**

```

[
  {
    "__KEY": 416,
    "__STAMP": 1,
    "ID": 416,
    "firstName": "Gregg",
    "lastName": "Wahl",
    "salary": 79100,
    "birthDate": "1963-02-01T00:00:00.000Z",
    "woman": false,
    "managerID": 412,
    "employerID": 20,
    "photo": "[object Picture]",
    "extra": null,
    "employer": {
      "__KEY": 20
    },
    "manager": {
      "__KEY": 412
    }
  },
  {
    "__KEY": 417,
    "__STAMP": 1,
    "ID": 417,
    "firstName": "Irma",
    "lastName": "Durham",
    "salary": 47000,
    "birthDate": "1992-06-16T00:00:00.000Z",
    "woman": true,
    "managerID": 412,
    "employerID": 20,
    "photo": "[object Picture]",
    "extra": null,
    "employer": {
      "__KEY": 20
    },
    "manager": {
      "__KEY": 412
    }
  }
]

```

### Example 3

Example with slicing and filtering on properties:

```

var $employeesCollection; $filter : Collection
var $employees : cs.EmployeeSelection

$employeesCollection:=New collection
$filter:=New collection

$filter.push("firstName")
$filter.push("lastName")

$employees:=ds.Employee.all()
$employeesCollection:=$employees.toCollection($filter;0;0;2)

```

Returns:

```
[
  {
    "firstName": "Gregg",
    "lastName": "Wahl"
  },
  {
    "firstName": "Irma",
    "lastName": "Durham"
  }
]
```

## Example 4

Example with `relatedEntity` type with simple form:

```
var $employeesCollection : Collection
$employeesCollection:=New collection
$employeesCollection:=$employees.toCollection("firstName,lastName,emplo
```

returns:

```
[
  {
    "firstName": "Gregg",
    "lastName": "Wahl",
    "employer": {
      "__KEY": 20
    }
  },
  {
    "firstName": "Irma",
    "lastName": "Durham",
    "employer": {
      "__KEY": 20
    }
  },
  {
    "firstName": "Lorena",
    "lastName": "Boothe",
    "employer": {
      "__KEY": 20
    }
  }
]
```

## Example 5

Example with `filterCol` parameter:

```
var $employeesCollection; $coll : Collection
$employeesCollection:=New collection
$coll:=New collection("firstName";"lastName")
$employeesCollection:=$employees.toCollection($coll)
```

Returns:

```
[
  {
    "firstName": "Joanna",
    "lastName": "Cabrera"
  },
  {
    "firstName": "Alexandra",
    "lastName": "Coleman"
  }
]
```

## Example 6

Example with extraction of all properties of a relatedEntity:

```
var $employeesCollection; $coll : Collection
$employeesCollection:=New collection
$coll:=New collection
$coll.push("firstName")
$coll.push("lastName")
$coll.push("employer.*")
$employeesCollection:=$employees.toCollection($coll)
```

Returns:

```
[
  {
    "firstName": "Gregg",
    "lastName": "Wahl",
    "employer": {
      "ID": 20,
      "name": "India Astral Secretary",
      "creationDate": "1984-08-25T00:00:00.000Z",
      "revenues": 12000000,
      "extra": null
    }
  },
  {
    "firstName": "Irma",
    "lastName": "Durham",
    "employer": {
      "ID": 20,
      "name": "India Astral Secretary",
      "creationDate": "1984-08-25T00:00:00.000Z",
      "revenues": 12000000,
      "extra": null
    }
  },
  {
    "firstName": "Lorena",
    "lastName": "Boothe",
    "employer": {
      "ID": 20,
      "name": "India Astral Secretary",
      "creationDate": "1984-08-25T00:00:00.000Z",
      "revenues": 12000000,
      "extra": null
    }
  }
]
```

## Example 7

Example with extraction of some properties of a relatedEntity:

```

var $employeesCollection : Collection
$employeesCollection:=New collection
$employeesCollection:=$employees.toCollection("firstName, lastName,
employer.name")
[
  {
    "firstName": "Gregg",
    "lastName": "Wahl",

    "employer": {
      "name": "India Astral Secretary"
    }
  },
  {
    "firstName": "Irma",
    "lastName": "Durham",
    "employer": {
      "name": "India Astral Secretary"
    }
  },
  {
    "firstName": "Lorena",
    "lastName": "Boothe",
    "employer": {
      "name": "India Astral Secretary"
    }
  }
]

```

## Example 8

Example with extraction of some properties of `relatedEntities`:

```

var $employeesCollection : Collection
$employeesCollection:=New collection
$employeesCollection:=$employees.toCollection("firstName, lastName,
directReports.firstName")

```

Returns:

```
[
  {
    "firstName": "Gregg",
    "lastName": "Wahl",
    "directReports": []
  },
  {
    "firstName": "Mike",
    "lastName": "Phan",
    "directReports": [
      {
        "firstName": "Gary"
      },
      {
        "firstName": "Sadie"
      },
      {
        "firstName": "Christie"
      }
    ]
  },
  {
    "firstName": "Gary",
    "lastName": "Reichert",
    "directReports": [
      {
        "firstName": "Rex"
      },
      {
        "firstName": "Jenny"
      },
      {
        "firstName": "Lowell"
      }
    ]
  }
]
```

## Example 9

Example with extraction of all properties of `relatedEntities`:

```
var $employeesCollection : Collection
$employeesCollection:=New collection
$employeesCollection:=$employees.toCollection("firstName, lastName,
directReports.*")

[
  {
    "firstName": "Gregg",
    "lastName": "Wahl",
    "directReports": []
  },
  {
    "firstName": "Mike",
    "lastName": "Phan",
    "directReports": [
      {
        "ID": 425,
        "firstName": "Gary",
        "lastName": "Reichert",
        "salary": 65800,
        "birthDate": "1957-12-23T00:00:00.000Z",
        "woman": false,
        "managerID": 424,
        "reports": [
          {
            "ID": 426,
            "firstName": "Sadie",
            "lastName": "Christie",
            "salary": 55000,
            "birthDate": "1962-05-15T00:00:00.000Z",
            "woman": true,
            "managerID": 425
          }
        ]
      }
    ]
  }
]
```

```
        "employerID": 21,
        "photo": "[object Picture]",
        "extra": null,
        "employer": {
            "__KEY": 21
        },
        "manager": {
            "__KEY": 424
        }
    },
    {
        "ID": 426,
        "firstName": "Sadie",
        "lastName": "Gallant",
        "salary": 35200,
        "birthDate": "2022-01-03T00:00:00.000Z",
        "woman": true,
        "managerID": 424,
        "employerID": 21,
        "photo": "[object Picture]",
        "extra": null,
        "employer": {
            "__KEY": 21
        },
        "manager": {
            "__KEY": 424
        }
    }
]
},
{
    "firstName": "Gary",
    "lastName": "Reichert",
    "directReports": [
        {
            "ID": 428,
            "firstName": "Rex",
            "lastName": "Chance",
            "salary": 71600,
            "birthDate": "1968-08-09T00:00:00.000Z",
            "woman": false,

            "managerID": 425,
            "employerID": 21,
            "photo": "[object Picture]",
            "extra": null,
            "employer": {
                "__KEY": 21
            },
            "manager": {
                "__KEY": 425
            }
        },
        {
            "ID": 429,
            "firstName": "Jenny",
            "lastName": "Parks",
            "salary": 51300,
            "birthDate": "1984-05-25T00:00:00.000Z",
            "woman": true,
            "managerID": 425,
            "employerID": 21,
            "photo": "[object Picture]",
            "extra": null,
            "employer": {
                "__KEY": 21
            }
        }
    ]
}
```

```
        " __KEY": 21
    },
    "manager": {
        " __KEY": 425
    }
}
]
```

# File

`File` objects are created with the [File](#) command. They contain references to disk files that may or may not actually exist on disk. For example, when you execute the `File` command to create a new file, a valid `File` object is created but nothing is actually stored on disk until you call the [`file.create\(\)`](#) function.

## Example

The following example creates a preferences file in the project folder:

```
var $created : Boolean  
$created:=File("/PACKAGE/SpecialPrefs/"+Current  
user+".myPrefs").create()
```

## Pathnames

`File` objects support several pathnames, including [filesystems](#) or [posix](#) syntax. Supported pathnames are detailed in the [Pathnames](#) page.

## File object

**.copyTo( destinationFolder : 4D.Folder { ; newName : Text } { ; overwrite : Integer } ) : 4D.File** copies the `File` object into the specified `destinationFolder`

**.create() : Boolean** creates a file on disk according to the properties of the `File` object

**.createAlias( destinationFolder : 4D.Folder ; aliasName : Text { ; aliasType : Integer } ) : 4D.File** creates an alias (macOS) or a shortcut (Windows)

**.creationDate : Date** the creation date of the file

**.creationTime : Time** the creation time of the file

**.delete()** deletes the file

**.exists : Boolean** true if the file exists on disk

**.extension : Text** the extension of the file name (if any)

**.fullName : Text** the full name of the file, including its extension (if any)

**.getAppInfo() : Object** returns the contents of a `.exe`, `.dll` or `.plist` file information as an object

**.getContent( ) : 4D.Blob** returns a `4D.Blob` object containing the entire content of a file

**.getIcon( { size : Integer } ) : Picture** the icon of the file

**.getText( { charSetName : Text { ; breakMode : Integer } } ) : Text**

**.getText( { charSetNum : Integer { ; breakMode : Integer } } ) : Text** returns the contents of the file as text

**.hidden : Boolean** true if the file is set as "hidden" at the system level

**.isAlias : Boolean** true if the file is an alias, a shortcut, or a symbolic link

**.isFile : Boolean** always true for a file

**.isFolder : Boolean** always false for a file

**.isWritable : Boolean** true if the file exists on disk and is writable

**.modificationDate : Date** the date of the file's last modification

**.modificationTime : Time** the time of the file's last modification

**.moveTo( destinationFolder : 4D.Folder { ; newName : Text } ) : 4D.File** moves or renames the `File` object into the specified `destinationFolder`

**.name : Text** the name of the file without extension (if any)

**.open( { mode : Text } ) : 4D.FileHandle**

**.open( { options : Object } ) : 4D.FileHandle** creates and returns a new `4D.FileHandle` object on the file, in the specified `mode` or with the specified `options`

**.original : 4D.File**

**.original : 4D.Folder** the target element for an alias, a shortcut, or a symbolic link file

**.parent : 4D.Folder** the parent folder object of the file

**.path : Text** the POSIX path of the file

**.platformPath : Text** the path of the file expressed with the current platform syntax

**.rename( newName : Text ) : 4D.File** renames the file with the name you passed in `newName` and returns the renamed `File` object

**.setAppInfo( info : Object )** writes the `info` properties as information contents of a `.exe`, `.dll` or `.plist` file

**.setContent ( content : Blob )** rewrites the entire content of the file using the data stored in the `content` BLOB

**.setText ( text : Text {; charSetName : Text { ; breakMode : Integer } } )**

**.setText ( text : Text {; charSetNum : Integer { ; breakMode : Integer } } )** writes `text` as the new contents of the file

**.size : Real** the size of the file expressed in bytes

## File

- History

**File** ( *path* : Text { ; *pathType* : Integer }{ ; } ) : 4D.File  
**File** ( *fileConstant* : Integer { ; } ) : 4D.File

| Parameter           | Type    | Description  |
|---------------------|---------|--|
| <i>path</i>         | Text    | -> File path   |
| <i>fileConstant</i> | Integer | -> 4D file constant  |
| <i>pathType</i>     | Integer | -> <code>fk posix path</code> (default) or <code>fk platform path</code><br>* -> * to return file of host database |
| Result              | 4D.File | <- New file object   |

## Description

The `File` command creates and returns a new object of the `4D.File` type. The command accepts two syntaxes:

### **File ( *path* { ; *pathType* } { ; \* } )**

In the *path* parameter, pass a file path string. You can use a custom string or a filesystem (e.g., "/DATA/myfile.txt").

Only absolute pathnames are supported with the `File` command.

By default, 4D expects a path expressed with the POSIX syntax. If you work with platform pathnames (Windows or macOS), you must declare it using the *pathType* parameter. The following constants are available:

| Constant                      | Value | Comment   |
|-------------------------------|-------|---|
| <code>fk platform path</code> | 1     | Path expressed with a platform-specific syntax (mandatory in case of platform pathname) |
| <code>fk posix path</code>    | 0     | Path expressed with POSIX syntax (default)  |

### **File ( *fileConstant* { ; \* } )**

In the *fileConstant* parameter, pass a 4D built-in or system file, using one of the following constants:

| Constant                                     | Value | Comment   |
|--|-------|---|
| <code>Backup history file</code>             | 19    | Backup history file (see Configuration and trace files). Stored in the backup destination folder.   |
| <code>Backup log file</code>                 | 13    | Current backup journal file. Stored in the application Logs folder.   |
| <code>Backup settings file</code>            | 1     | Default backup.4DSettings file (xml format), stored in the Settings folder of the project   |
| <code>Backup settings file for data</code>   | 17    | backup.4DSettings file (xml format) for the data file, stored in the Settings folder of the data folder   |
| <code>Build application log file</code>      | 14    | Current log file in xml format of the application builder. Stored in the Logs folder.   |
| <code>Build application settings file</code> | 20    | Default settings file of the application builder ("buildApp.4DSettings"). Stored in the Settings folder of the project.                           |
| <code>Compacting log file</code>             | 6     | Log file of the most recent compacting done with the Compact data file command or the Maintenance and security center. Stored in the Logs folder. |
| Current                                      |       | backup.4DSettings file currently used by the application. It can be   |

|                                   |                                    |  |
|-----------------------------------|------------------------------------|--|
| <b>constantValue</b>              | the backup settings file (default) | Custom user backup settings file defined for the data file   |
| Debug log file                    | 12                                 | Log file created by the <code>SET DATABASE PARAMETER(Debug log recording)</code> command. Stored in the Logs folder.   |
| Diagnostic log file               | 11                                 | Log file created by the <code>SET DATABASE PARAMETER(Diagnostic log recording)</code> command. Stored in the Logs folder.  |
| Directory file                    | 16                                 | directory.json file, containing the description of users and groups (if any) for the project application. It can be located either in the user settings folder (default, global to the project), or in the data settings folder (specific to a data file).   |
| HTTP debug log file               | 9                                  | Log file created by the <code>WEB SET OPTION(Web debug log)</code> command. Stored in the Logs folder.   |
| HTTP log file                     | 8                                  | Log file created by the <code>WEB SET OPTION(Web log recording)</code> command. Stored in Logs folder.   |
| IMAP Log file                     | 23                                 | Log file created by the <code>SET DATABASE PARAMETER(IMAP Log)</code> command. Stored in the Logs folder.  |
| Last backup file                  | 2                                  | Last backup file, named <code>\&lt;applicationName&gt;[bkpNum].4BK</code> , stored at a custom location.   |
| Last journal integration log file | 22                                 | Full pathname of the last journal integration log file (stored in the Logs folder of the restored application), if any. This file is created, in auto-repair mode, as soon as a log file integration occurred  |
| Repair log file                   | 7                                  | Log file of database repairs made on the database in the Maintenance and Security Center (MSC). Stored in the Logs folder.   |
| Request log file                  | 10                                 | Standard client/server request log file (excluding Web requests) created by the <code>SET DATABASE PARAMETER(4D Server log recording)</code> or <code>SET DATABASE PARAMETER(Client log recording)</code> commands. If executed on the server, the server log file is returned (stored in the Logs folder on the server). If executed on the client, the client log file is returned (stored in the client local Logs folder). |
| SMTP log file                     | 15                                 | Log file created by the <code>SET DATABASE PARAMETER(SMTP Log)</code> command. Stored in the Logs folder.  |
| User settings file                | 3                                  | settings.4DSettings file for all data files, stored in Preferences folder next to structure file if enabled.   |
| User settings file for data       | 4                                  | settings.4DSettings file for current data file, stored in Preferences folder next to the data file.  |
| Verification log file             | 5                                  | Log files created by the <code>VERIFY CURRENT DATA FILE</code> and <code>VERIFY DATA FILE</code> commands or the Maintenance and Security Center (MSC). Stored in the Logs folder.   |

If the target `fileConstant` does not exist, a null object is returned. No errors are raised.

If the command is called from a component, pass the optional `*` parameter to get the path of the host database. Otherwise, if you omit the `*` parameter, a null object is always returned.

## 4D.File.new()

- History

**4D.File.new** ( `path : Text { ; pathType : Integer }{ ; }` ) : `4D.File`  
**4D.File.new** ( `fileConstant : Integer { ; }` ) : `4D.File`

## Description

The `4D.File.new()` function creates and returns a new object of the `4D.File` type. It is identical to the [File](#) command (shortcut).

It is recommended to use the [File](#) shortcut command instead of `4D.File.new()`.

## .copyTo()

- History

**.copyTo(** *destinationFolder* : 4D.Folder { ; *newName* : Text } { ; *overwrite* : Integer }  
**)** : 4D.File

| Parameter                | Type        | Description   |
|--------------------------|-------------|---|
| <i>destinationFolder</i> | 4D.Folder-> | Destination folder  |
| <i>newName</i>           | Text        | -> Name for the copy                                      |
| <i>overwrite</i>         | Integer     | -> <code>fk overwrite</code> to replace existing elements |
| Result                   | 4D.File     | <- Copied file  |

## Description

The `.copyTo()` function copies the `File` object into the specified *destinationFolder*.

The *destinationFolder* must exist on disk, otherwise an error is generated.

By default, the file is copied with the name of the original file. If you want to rename the copy, pass the new name in the *newName* parameter. The new name must comply with naming rules (e.g., it must not contain characters such as ":", "/", etc.), otherwise an error is returned.

If a file with the same name already exists in the *destinationFolder*, by default 4D generates an error. You can pass the `fk overwrite` constant in the *overwrite* parameter to ignore and overwrite the existing file

| Constant                  | Value    | Comment                             |
|---------------------------|----------|-------------------------------------|
| <code>fk overwrite</code> | 4<br>any | Overwrite existing elements, if any |

## Returned value

The copied `File` object.

## Example

You want to copy a picture *file* from the user's document folder to the application folder:

```
var $source; $copy : Object
$source:=Folder(fk documents folder).file("Pictures/photo.png")
$copy:=$source.copyTo(Folder("/PACKAGE") ; fk overwrite)
```

## .create()

- History

## Not available for ZIP archives

## .create() : Boolean

| Parameter | Type    | Description   |
|-----------|---------|---|
| Result    | Boolean | <- True if the file was created successfully, false otherwise |

### Description

The `.create()` function creates a file on disk according to the properties of the `File` object.

If necessary, the function creates the folder hierarchy as described in the [platformPath](#) or [path](#) properties. If the file already exists on disk, the function does nothing (no error is thrown) and returns false.

### Returned value

- **True** if the file is created successfully;
- **False** if a file with the same name already exists or if an error occurred.

### Example

Creation of a preferences file in the database folder:

```
var $created : Boolean  
$created:=File("/PACKAGE/SpecialPrefs/"+Current  
user+".myPrefs").create()
```

## .createAlias()

- History

**.createAlias(** *destinationFolder* : 4D.Folder ; *aliasName* : Text { ; *aliasType* : Integer } **)** : 4D.File

| Parameter                | Type      | Description                                     |
|--------------------------|-----------|---|
| <i>destinationFolder</i> | 4D.Folder | <- Destination folder for the alias or shortcut |
| <i>aliasName</i>         | Text      | -> Name of the alias or shortcut                |
| <i>aliasType</i>         | Integer   | -> Type of the alias link                       |
| Result                   | 4D.File   | <- Alias or shortcut file reference             |

### Description

The `.createAlias()` function creates an alias (macOS) or a shortcut (Windows) to the file with the specified *aliasName* name in the folder designated by the *destinationFolder* object.

Pass the name of the alias or shortcut to create in the *aliasName* parameter.

By default on macOS, the function creates a standard alias. You can also create a symbolic link by using the *aliasType* parameter. The following constants are available:

| Constant                      | Value | Comment                    |
|-------------------------------|-------|----------------------------|
| <code>fk alias link</code>    | 0     | Alias link (default)       |
| <code>fk symbolic link</code> | 1     | Symbolic link (macOS only) |

On Windows, a shortcut (.lnk file) is always created (the *aliasType* parameter is ignored).

## Returned object

A `4D.File` object with the `isAlias` property set to **true**.

## Example

You want to create an alias to a file in your database folder:

```
$myFile:=Folder(fk documents folder).file("Archives/ReadMe.txt")
$aliasFile:=$myFile.createAlias(File("/PACKAGE");"ReadMe")
```

## .creationDate

- History

**.creationDate** : Date

### Description

The `.creationDate` property returns the creation date of the file.

This property is **read-only**.

## .creationTime

- History

**.creationTime** : Time

### Description

The `.creationTime` property returns the creation time of the file (expressed as a number of seconds beginning at 00:00).

This property is **read-only**.

## .delete()

- History

**.delete()**

| Parameter Type | Description                     |
|----------------|---------------------------------|
|                | Does not require any parameters |

### Description

The `.delete()` function deletes the file.

If the file is currently open, an error is generated.

If the file does not exist on disk, the function does nothing (no error is generated).

**WARNING:** `.delete()` can delete any file on a disk. This includes documents created with other applications, as well as the applications

themselves. `.delete()` should be used with extreme caution. Deleting a file is a permanent operation and cannot be undone.

## Example

You want to delete a specific file in the database folder:

```
$tempo:=File("/PACKAGE/SpecialPrefs/"+Current user+".prefs")
If($tempo.exists)
    $tempo.delete()
    ALERT("User preference file deleted.")
End if
```

## .exists

- History

**.exists** : Boolean

### Description

The `.exists` property returns true if the file exists on disk, and false otherwise.

This property is **read-only**.

## .extension

- History

**.extension** : Text

### Description

The `.extension` property returns the extension of the file name (if any). An extension always starts with `".`. The property returns an empty string if the file name does not have an extension.

This property is **read-only**.

## .fullName

- History

**.fullName** : Text

### Description

The `.fullName` property returns the full name of the file, including its extension (if any).

This property is **read-only**.

## .getAppInfo()

- History

**.getAppInfo()** : Object

| Parameter Type        | Description   |
|-----------------------|---|
| Result      Object <- | Contents of .exe/.dll version resource or .plist file |

## Description

The `.getAppInfo()` function returns the contents of a **.exe**, **.dll** or **.plist** file information as an object.

The function must be used with an existing .exe, .dll or .plist file. If the file does not exist on disk or is not a valid .exe, .dll or .plist file, the function returns an empty object (no error is generated).

The function only supports .plist files in xml format (text-based). An error is returned if it is used with a .plist file in binary format.

### Returned object with a .exe or .dll file

Reading a .exe or .dll is only possible on Windows.

All property values are Text.

| Property         | Type |
|------------------|------|
| InternalName     | Text |
| ProductName      | Text |
| CompanyName      | Text |
| LegalCopyright   | Text |
| ProductVersion   | Text |
| FileDescription  | Text |
| FileVersion      | Text |
| OriginalFilename | Text |

### Returned object with a .plist file

The xml file contents is parsed and keys are returned as properties of the object, preserving their types (text, boolean, number). `.plist dict` is returned as a JSON object and `.plist array` is returned as a JSON array.

## Example

```
// display copyright info of application .exe file (windows)
var $exeFile : 4D.File
var $info : Object
$exeFile:=File(Application file; fk platform path)
$info:=$exeFile.getAppInfo()
ALERT($info.LegalCopyright)

// display copyright info of an info.plist (any platform)
var $infoPlistFile : 4D.File
var $info : Object
$infoPlistFile:=File("/RESOURCES/info.plist")
$info:=$infoPlistFile.getAppInfo()
ALERT($info.Copyright)
```

## See also

[.setAppInfo\(\)](#)

## .getContent()

- History

**.getContent( )** : 4D.Blob

### Parameter Type Description

Result 4D.Blob <- File content

### Description

The `.getContent()` function returns a `4D.Blob` object containing the entire content of a file. For information on BLOBs, please refer to the [BLOB](#) section.

### Returned value

A `4D.Blob` object.

### Example

To save a document's contents in a `BLOB` field:

```
var $vPath : Text
$vPath:=Select document(""); "*" ; "Select a document"; 0
If (OK=1) //If a document has been chosen
    [aTable]aBlobField:=File($vPath;fk platform path).getContent()
End if
```

## .getIcon()

- History

**.getIcon( { size : Integer } )** : Picture

### Parameter Type Description

size Integer-> Side length for the returned picture  
(pixels)

Result Picture <- Icon

### Description

The `.getIcon()` function returns the icon of the file.

The optional `size` parameter specifies the dimensions in pixels of the returned icon. This value actually represents the length of the side of the square containing the icon. Icons are usually defined in 32x32 pixels ("large icons") or 16x16 pixels ("small icons"). If you pass 0 or omit this parameter, the "large icon" version is returned.

If the file does not exist on disk, a default blank icon is returned.

### Returned value

File icon [picture](#).

## .getText()

- History

**.getText( { charsetName : Text { ; breakMode : Integer } } )** : Text

**.getText( { charSetName : Integer { ; breakMode : Integer } } ) : Text**

| <b>Parameter</b> | <b>Type</b> | <b>Description</b>                 |
|------------------|-------------|------------------------------------|
| charSetName      | Text        | -> Name of character set           |
| charSetNum       | Integer     | -> Number of character set         |
| breakMode        | Integer     | -> Processing mode for line breaks |
| Result           | Text        | <- Text from the document          |

## Description

The `.getText()` function returns the contents of the file as text .

Optionally, you can designate the character set to be used for reading the contents. You can pass either:

- in `charSetName`, a string containing the standard set name (for example "ISO-8859-1" or "UTF-8"),
- or in `charSetNum`, the MIBEnum ID (number) of the standard set name.

For the list of character sets supported by 4D, refer to the description of the `CONVERT FROM TEXT` command.

If the document contains a Byte Order Mark (BOM), 4D uses the character set that it has set instead of the one specified in `charSetName` or `charSetNum` (this parameter is then ignored). If the document does not contain a BOM and if `charSetName` or `charSetNum` is omitted, by default 4D uses the "UTF-8" character set.

In `breakMode`, you can pass a number indicating the processing to apply to end-of-line characters in the document. The following constants of the "System Documents" theme are available:

| <b>Constant</b>             | <b>Value</b> | <b>Comment</b>  |
|-----------------------------|--------------|---|
| Document unchanged          | 0            | No processing   |
| Document with native format | 1            | (Default) Line breaks are converted to the native format of the operating system: CR (carriage return) under OS X, CRLF (carriage return + line feed) under Windows |
| Document with CRLF          | 2            | Line breaks are converted to Windows format: CRLF (carriage return + line feed)   |
| Document with CR            | 3            | Line breaks are converted to OS X format: CR (carriage return)  |
| Document with LF            | 4            | Line breaks are converted to Unix format: LF (line feed)  |

By default, when you omit the `breakMode` parameter, line breaks are processed in native mode (1).

## Returned value

Text of the file.

## Example

Given the following text document (fields are separated by tabs):

```
id name price vat
3 thé 1.06€ 19.6
2 café 1.05€ 19.6
```

When you execute this code:

```
$myFile:=Folder(fk documents folder).file("Billing.txt") //UTF-8 by
default
$txt:=$myFile.getText()
```

... you get the following for `$txt`:

```
"id\tname\tprice\tvat\r\n3\tthé\t1.06€\t19.6\r\n2\tcafé\t1.05€\t19.6"
```

with `\t` (tab) as separator and `\r\n` (CRLF) as line delimiter.

Here is another example with the same file, but a different line delimiter:

```
$txt:=$myFile.getText("UTF-8"; Document with LF)
```

In this case, the contents of `$txt` are as follows:

```
"id\tname\tprice\tvat\n3\tthé\t1.06€\t19.6\n2\tcafé\t1.05€\t19.6"
```

This time `\n` (LF) is used as line delimiter.

## .hidden

- History

**.hidden** : Boolean

### Description

The `.hidden` property returns true if the file is set as "hidden" at the system level, and false otherwise.

This property is **read/write**.

## .isAlias

- History

**.isAlias** : Boolean

### Description

The `.isAlias` property returns true if the file is an alias, a shortcut, or a symbolic link, and false otherwise.

This property is **read-only**.

## .isFile

- History

**.isFile** : Boolean

### Description

The `.isFile` property returns always true for a file.

This property is **read-only**.

## **.isFolder**

- History

**.isFolder** : Boolean

### **Description**

The `.isFolder` property returns always false for a file.

This property is **read-only**.

## **.isWritable**

- History

**.isWritable** : Boolean

### **Description**

The `.isWritable` property returns true if the file exists on disk and is writable.

The property checks the ability of the 4D application to write on the disk (access rights), it does not solely rely on the *writable* attribute of the file.

This property is **read-only**.

### **Example**

```
$myFile:=File("C:\\\\Documents\\\\Archives\\\\ReadMe.txt";fk platform path)
If ($myFile.isWritable)
    $myNewFile:=$myFile.setText("Added text")
End if
```

## **.modificationDate**

- History

**.modificationDate** : Date

### **Description**

The `.modificationDate` property returns the date of the file's last modification.

This property is **read-only**.

## **.modificationTime**

- History

**.modificationTime** : Time

### **Description**

The `.modificationTime` property returns the time of the file's last modification

(expressed as a number of seconds beginning at 00:00).

This property is **read-only**.

## .moveTo()

- History

**.moveTo( *destinationFolder* : 4D.Folder { ; *newName* : Text } ) : 4D.File**

| Parameter                | Type        | Description                     |
|--------------------------|-------------|---------------------------------|
| <i>destinationFolder</i> | 4D.Folder-> | Destination folder              |
| <i>newName</i>           | Text        | -> Full name for the moved file |
| Result                   | 4D.File     | <- Moved file                   |

### Description

The `.moveTo()` function moves or renames the `File` object into the specified *destinationFolder*.

The *destinationFolder* must exist on disk, otherwise an error is generated.

By default, the file retains its name when moved. If you want to rename the moved file, pass the new full name in the *newName* parameter. The new name must comply with naming rules (e.g., it must not contain characters such as ":", "/", etc.), otherwise an error is returned.

### Returned object

The moved `File` object.

### Example

```
$DocFolder:=Folder(fk documents folder)
$myFile:=$DocFolder.file("Current/Infos.txt")
$myFile.moveTo($DocFolder.folder("Archives");"Infos_old.txt")
```

## .name

- History

**.name** : Text

### Description

The `.name` property returns the name of the file without extension (if any).

This property is **read-only**.

## .open()

- History

**.open( { mode : Text } ) : 4D.FileHandle**

**.open( { options : Object } ) : 4D.FileHandle**

| Parameter | Type                          | Description                                |
|-----------|-------------------------------|--|
| mode      | Text                          | -> Opening mode: "read", "write", "append" |
| options   | Object                        | -> Opening options                         |
| Result    | <a href="#">4D.FileHandle</a> | <- New File handle object                  |

## Description

The `.open()` function creates and returns a new [4D.FileHandle](#) object on the file, in the specified *mode* or with the specified *options*. You can use functions and properties of the [4D.FileHandle](#) class to write, read, or append contents to the file.

If you use the *mode* (text) parameter, pass the opening mode for the file handle:

| <b>mode</b> | <b>Description</b>   |
|-------------|--|
| "read"      | (Default) Creates a file handle to read values from the file. If the file does not exist on disk, an error is returned. You can open as many file handles as you want in "read" mode on the same File object.                    |
| "write"     | Creates a file handle to write values to the file (starting at the beginning of the file content). If the file does not exist on disk, it is created. You can open only one file handle in "write" mode on the same File object. |
| "append"    | Creates a file handle to write values to the file (starting at the end of the file content). If the file does not exist on disk, it is created. You can open only one file handle in "append" mode on the same File object.      |

The *mode* value is case sensitive.

If you use the *options* (object) parameter, you can pass more options for the file handle through the following properties (these properties can be read afterwards from the opened [file handle object](#)):

| <b>options</b>               | <b>Type</b>    | <b>Description</b>  | <b>Default</b> |
|------------------------------|----------------|---|----------------|
| <code>.mode</code>           | Text           | Opening mode (see <i>mode</i> above)  | "read"         |
| <code>.charset</code>        | Text           | Charset used when reading from or writing to the file. Use the standard name of the set (for example "UTF-8" "ISO-8859-1" or "UTF-8") |                |
| <code>.breakModeRead</code>  | Text or Number | Processing mode for line breaks used when reading "native" Number in the file (see below)   | or 1           |
| <code>.breakModeWrite</code> | Text or Number | Processing mode for line breaks used when writing "native" Number to the file (see below)   | or 1           |

The `.breakModeRead` and `.breakModeWrite` indicate the processing to apply to end-of-line characters in the document. You can use one of the following values (text or number):

| <b>Break Break mode</b>     | <b>Description</b>  |
|-----------------------------|---|
| <b>mode as number</b>       |   |
| <b>as text (constant)</b>   |   |
| "native" with native format | 1 (Document) (Default) Line breaks are converted to the native format of the operating system: LF (line feed) under macOS, CRLF (carriage return + line feed) under Windows |
| "crlf"                      | 2 (Document with CRLF) Line breaks are converted to CRLF (carriage return + line feed), the default Windows format  |
| "cr"                        | 3 (Document with CR) Line breaks are converted to CR (carriage return), the default Classic Mac OS format   |
| "lf"                        | 4 (Document with LF) Line breaks are converted to LF (line feed), the default Unix and macOS format   |

The *break mode as text* value is case sensitive.

## Example

You want to create a file handle for reading the "ReadMe.txt" file:

```
var $f : 4D.File
var $fhandle : 4D.FileHandle

$f:=File("C:\\Documents\\Archives\\ReadMe.txt";fk platform path)
$fhandle:=$f.open("read")
```

## .original

- History

**.original** : 4D.File  
**.original** : 4D.Folder

## Description

The `.original` property returns the target element for an alias, a shortcut, or a symbolic link file. The target element can be:

- a file object
- a folder object

For non-alias files, the property returns the same file object as the file.

This property is **read-only**.

## .parent

- History

**.parent** : 4D.Folder

## Description

The `.parent` property returns the parent folder object of the file. If the path represents a system path (e.g., "/DATA/"), the system path is returned.

This property is **read-only**.

## .path

- History

**.path** : Text

### Description

The `.path` property returns the POSIX path of the file. If the path represents a filesystem (e.g., `"/DATA/"`), the filesystem is returned.

This property is **read-only**.

## .platformPath

- History

**.platformPath** : Text

### Description

The `.platformPath` property returns the path of the file expressed with the current platform syntax.

This property is **read-only**.

## .rename()

- History

**.rename( newName : Text )** : 4D.File

| Parameter | Type    | Description                   |
|-----------|---------|-------------------------------|
| newName   | Text    | -> New full name for the file |
| Result    | 4D.File | <- Renamed file               |

### Description

The `.rename()` function renames the file with the name you passed in `newName` and returns the renamed `File` object.

The `newName` parameter must comply with naming rules (e.g., it must not contain characters such as `:`, `/`, etc.), otherwise an error is returned. If a file with the same name already exists, an error is returned.

Note that the function modifies the full name of the file, i.e. if you do not pass an extension in `newName`, the file will have a name without an extension.

### Returned object

The renamed `File` object.

### Example

You want to rename "ReadMe.txt" in "ReadMe\_new.txt":

```

$toRename:=File("C:\\Documents\\Archives\\ReadMe.txt";fk platform
path)
$newName:=$toRename.rename($toRename.name+"_new"+$toRename.extension)

```

## .setAppInfo()

- History

**.setAppInfo( *info* : Object )**

| Parameter | Type         | Description   |
|-----------|--------------|---|
| info      | Object->file | Properties to write in .exe/.dll version resource or .plist |

### Description

The `.setAppInfo()` function writes the *info* properties as information contents of a **.exe**, **.dll** or **.plist** file.

The function must be used with an existing .exe, .dll or .plist file. If the file does not exist on disk or is not a valid .exe, .dll or .plist file, the function does nothing (no error is generated).

The function only supports .plist files in xml format (text-based). An error is returned if it is used with a .plist file in binary format.

#### ***info* parameter object with a .exe or .dll file**

Writing a .exe or .dll file information is only possible on Windows.

Each valid property set in the *info* object parameter is written in the version resource of the .exe or .dll file. Available properties are (any other property will be ignored):

| Property         | Type | Comment   |
|------------------|------|---|
| InternalName     | Text |   |
| ProductName      | Text |   |
| CompanyName      | Text |   |
| LegalCopyright   | Text |   |
| ProductVersion   | Text |   |
| FileDescription  | Text |   |
| FileVersion      | Text |   |
| OriginalFilename | Text |   |
| WinIcon          | Text | Posix path of .ico file. This property applies only to 4D generated executable files. |

For all properties except `WinIcon`, if you pass a null or empty text as value, an empty string is written in the property. If you pass a value type different from text, it is stringified.

For the `WinIcon` property, if the icon file does not exist or has an incorrect format, an error is generated.

#### ***info* parameter object with a .plist file**

Each valid property set in the *info* object parameter is written in the .plist file as a key. Any key name is accepted. Value types are preserved when possible.

If a key set in the *info* parameter is already defined in the .plist file, its value is

updated while keeping its original type. Other existing keys in the .plist file are left untouched.

To define a Date type value, the format to use is a json timestamp string formated in ISO UTC without milliseconds ("2003-02-01T01:02:03Z") like in the Xcode plist editor.

## Example

```
// set copyright, version and icon of a .exe file (Windows)
var $exeFile; $iconFile : 4D.File
var $info : Object
$exeFile:=File(Application file; fk platform path)
$iconFile:=File("/RESOURCES/myApp.ico")
$info:=New object
$info.LegalCopyright:="Copyright 4D 2023"
$info.ProductVersion:="1.0.0"
$info.WinIcon:=$iconFile.path
$exeFile.setAppInfo($info)

// set some keys in an info.plist file (all platforms)
var $infoPlistFile : 4D.File
var $info : Object
$infoPlistFile:=File("/RESOURCES/info.plist")
$info:=New object
$info.Copyright:="Copyright 4D 2023" //text
$info.ProductVersion:=12 //integer
$info.ShipmentDate:="2023-04-22T06:00:00Z" //timestamp
$info.CFBundleIconFile:="myApp.icns" //for macOS
$infoPlistFile.setAppInfo($info)
```

## See also

[.getAppInfo\(\)](#)

## .setContent()

- History

**.setContent ( content : Blob )**

| Parameter Type      | Description               |
|---------------------|---------------------------|
| content      BLOB-> | New contents for the file |

## Description

The `.setContent( )` function rewrites the entire content of the file using the data stored in the *content* BLOB. For information on BLOBS, please refer to the [BLOB](#) section.

## Example

```
$myFile:=Folder(fk documents folder).file("Archives/data.txt")
$myFile.setContent([aTable]aBlobField)
```

## .setText()

- History

**.setText ( text : Text {; charsetName : Text { ; breakMode : Integer } } )**

**.setText ( *text* : Text {; *charSetName* : Integer { ; *breakMode* : Integer } } )**

| Parameter          | Type    | Description                        |
|--------------------|---------|------------------------------------|
| <i>text</i>        | Text    | -> Text to store in the file       |
| <i>charSetName</i> | Text    | -> Name of character set           |
| <i>charSetNum</i>  | Integer | -> Number of character set         |
| <i>breakMode</i>   | Integer | -> Processing mode for line breaks |

## Description

The `.setText()` function writes *text* as the new contents of the file.

If the file referenced in the `File` object does not exist on the disk, it is created by the function. When the file already exists on the disk, its prior contents are erased, except if it is already open, in which case, its contents are locked and an error is generated.

In *text*, pass the text to write to the file. It can be a literal ("my text"), or a 4D text field or variable.

Optionally, you can designate the character set to be used for writing the contents. You can pass either:

- in *charSetName*, a string containing the standard set name (for example "ISO-8859-1" or "UTF-8"),
- or in *charSetNum*, the MIBEnum ID (number) of the standard set name.

For the list of character sets supported by 4D, refer to the description of the `CONVERT FROM TEXT` command.

If a Byte Order Mark (BOM) exists for the character set, 4D inserts it into the file unless the character set used contains the suffix "-no-bom" (e.g. "UTF-8-no-bom"). If you do not specify a character set, by default 4D uses the "UTF-8" character set without BOM.

In *breakMode*, you can pass a number indicating the processing to apply to end-of-line characters before saving them in the file. The following constants, found in the **System Documents** theme, are available:

| Constant                                 | Value | Comment  |
|--|-------|--|
| <code>Document unchanged</code>          | 0     | No processing  |
| <code>Document with native format</code> | 1     | (Default) Line breaks are converted to the native format of the operating system: LF (line feed) on macOS, CRLF (carriage return + line feed) on Windows |
| <code>Document with CRLF</code>          | 2     | Line breaks are converted to CRLF (carriage return + line feed), the default Windows format  |
| <code>Document with CR</code>            | 3     | Line breaks are converted to CR (carriage return), the default Classic Mac OS format   |
| <code>Document with LF</code>            | 4     | Line breaks are converted to LF (line feed), the default Unix and macOS format   |

By default, when you omit the *breakMode* parameter, line breaks are processed in native mode (1).

**Compatibility Note:** Compatibility options are available for EOL and BOM management. See [Compatibility page](#) on doc.4d.com.

## **Example**

```
$myFile:=File("C:\\Documents\\\\Hello.txt";fk platform path)  
$myFile.setText("Hello world")
```

## **.size**

- History

**.size** : Real

## **Description**

The **.size** property returns the size of the file expressed in bytes. If the file does not exist on disk, the size is 0.

This property is **read-only**.

## FileHandle

The `FileHandle` class has functions that allow you to sequentially read from or append contents to an opened `File` object. A file handle can access any part of a document.

File handle objects are created with the `file.open()` function.

To read or write a whole document at once, you might consider using the `file.getText()` and `file.setText()` functions.

Thanks to the standard 4D object *refcounting*, a file handle is automatically deleted when it is no longer referenced and thus, the requested `File` object is automatically closed. Consequently, with file handles you don't need to worry about closing documents.

### Example

```
var $f : 4D.File
var $fhandle : 4D.FileHandle
$f:=Folder(Database folder).file("example.txt")

//Writing line by line from the start
$fhandle:=$f.open("write")
$text:="Hello World"
For ($line; 1; 4)
    $fhandle.writeLine($text+String($line))
End for

//Writing line by line from the end
$fhandle:=$f.open("append")
$text:="Hello New World!"
For ($line; 1; 4)
    $fhandle.writeLine($text+String($line))
End for

//Reading using a stop character and an object parameter
$o:=New object()
$o.mode:="read"
$o.charset:="UTF-8"
$o.breakModeRead:=Document with CRLF
$stopChar:="!"
$fhandle:=$f.open($o)
$text:=$fhandle.readText($stopChar)

//Reading line by line
$lines:=New collection
$fhandle:=$f.open("read")
While (Not($fhandle.eof()))
    $lines.push($fhandle.readLine())
End while
```

### FileHandle object

File handle objects cannot be shared.

**.breakModeRead : Text** the processing mode for line breaks used when reading the file

**.breakModeWrite : Text** the processing mode for line breaks used when writing to the file

**.charset : Text** the charset used when reading from or writing to the file

**.eof : Boolean** True is the `offset` has reached the end of the file, and False otherwise

**.getSize() : Real** returns the current size of the document, expressed in bytes

**.mode : Text** the mode in which the file handle was created: "read", "write", or "append"

**.offset : Real** the current offset of the data stream (position inside the document)

**.readBlob( bytes : Real ) : [4D.Blob](BlobClass)** returns a blob a `bytes` size from the file, starting from the current position

**.readLine() : Text** returns a line of text from the current position until an end-of-line delimiter is encountered or the end of the document is reached

**.readText( { stopChar : Text } ) : Text** returns text from the file, starting from the current position until the first `stopChar` string is encountered (if passed) or the end of file is reached

**.setSize( size : Real )** sets a new `size` in bytes for the document

**.writeBlob( blob : 4D.Blob )** writes `blob` into the file, starting from the current position

**.writeLine( lineOfText : Text )** writes `lineOfText` content at the current position and inserts an end-of-line delimiter

**.writeText( textToWrite : Text )** writes `textToWrite` content at the current position and does not insert a final end-of-line delimiter

## .breakModeRead

- History

### .breakModeRead : Text

#### Description

The `.breakModeRead` property returns the processing mode for line breaks used when reading the file.

The `.breakModeRead` property can be defined at the handle creation with the `file.open()` function (see [the `.open\(\)` function](#) for more information). Default is "native".

The `.breakModeRead` property always contains a text value, even if the `.open()` option was set using a number (constant).

This property is **read-only**.

## .breakModeWrite

- History

### .breakModeWrite : Text

#### Description

The `.breakModeWrite` property returns the processing mode for line breaks used when writing to the file.

The `.breakModeWrite` property can be defined at the handle creation with the `file.open()` function (see [the `.open\(\)` function](#) for more information). Default is "native".

The `.breakModeWrite` property always contains a text value, even if the `.open()` option was set using a number (constant).

This property is **read-only**.

## .charset

- History

**.charset** : Text

### Description

The `.charset` property returns the charset used when reading from or writing to the file.

The charset can be defined at the handle creation with the `file.open()` function. Default is "UTF-8".

This property is **read-only**.

## .eof

- History

**.eof** : Boolean

### Description

The `.eof` property returns True if the `offset` has reached the end of the file, and False otherwise.

This property is **read-only**.

## .getSize()

- History

**.getSize()** : Real

| Parameter | Type | Description                   |
|-----------|------|-------------------------------|
| Result    | Real | Size of the document in bytes |

### Description

The `.getSize()` function returns the current size of the document, expressed in bytes.

This function returns the same value as the `(.size)` property of the `File` class.

## See also

[.setSize\(\)](#), [file.size](#)

## .mode

- History

**.mode** : Text

### Description

The `.mode` property returns the mode in which the file handle was created: "read", "write", or "append".

The mode can be defined at the handle creation with the [file.open\(\)](#) function. Default is "read".

This property is **read-only**.

## .offset

- History

**.offset** : Real

### Description

The `.offset` property returns the current offset of the data stream (position inside the document). The offset value is automatically updated after read and write operations.

Setting the `.offset` will change its current value.

- If the passed value is negative, the `.offset` is set to the start of the file (zero).
- If the passed value is higher than the size of the file, the `.offset` is set to the end of the file (size of file).

This property is **read/write**.

## .readBlob()

- History

**.readBlob( bytes : Real ) : 4D.Blob**

| Parameter | Type                    | Description                   |
|-----------|-------------------------|-------------------------------|
| bytes     | Real                    | -> Number of bytes to be read |
| Result    | <a href="#">4D.Blob</a> | <- Bytes read from the file   |

### Description

The `.readBlob()` function returns a blob a *bytes* size from the file, starting from the current position .

When this function is executed, the current position (`.offset`) is updated after the last byte read.

## See also

[.writeBlob\(\)](#)

## .readLine()

- History

[.readLine\(\)](#) : Text

### Parameter Type    Description

Result      Text <- Line of text

#### Description

The `.readLine()` function returns a line of text from the current position until an end-of-line delimiter is encountered or the end of the document is reached.

When this function is executed, the current position ([.offset](#)) is updated.

When this function is executed for the first time on a file handle, the whole document contents is loaded in a buffer.

#### See also

[.readText\(\)](#), [.writeLine\(\)](#)

## .readText()

- History

[.readText\( { stopChar : Text } \)](#) : Text

### Parameter Type                          Description

*stopChar*    Text -> Character(s) at which to stop reading

Result      Text <- Text from the file

#### Description

The `.readText()` function returns text from the file, starting from the current position until the first *stopChar* string is encountered (if passed) or the end of file is reached.

This function replaces all original end-of-line delimiters. By default, the native delimiter is used, but you can define another delimiter when [opening the file handle](#) by setting the [.breakModeRead](#) property.

The *stopChar* character string is not included in the returned text. If you omit the *stopChar* parameter, the whole document text is returned.

When this function is executed, the ([.offset](#)) is placed just after the *stopChar* string.

If the *stopChar* parameter is passed and not found, `.readText()` returns an empty string and the [.offset](#) is left untouched.

When this function is executed for the first time on a file handle, the whole document contents is loaded in a buffer.

#### See also

[.readLine\(\)](#), [.writeText\(\)](#)

## .setSize()

- History

**.setSize( size : Real )**

| Parameter Type    | Description                       |
|-------------------|-----------------------------------|
| size      Real -> | New size of the document in bytes |

### Description

The `.setSize()` function sets a new *size* in bytes for the document.

If the *size* value is less than the current document size, the document content is truncated from the beginning to get the new *size*.

### See also

[.getSize\(\)](#), [file.size](#)

## .writeBlob()

- History

**.writeBlob( blob : 4D.Blob )**

| Parameter Type                  | Description               |
|---------------------------------|---------------------------|
| blob <a href="#">4D.Blob</a> -> | Blob to write in the file |

### Description

The `.writeBlob()` function writes *blob* into the file, starting from the current position .

When this function is executed, the current position ([.offset](#)) is updated after the last byte written.

### See also

[.readBlob\(\)](#)

## .writeLine()

- History

**.writeLine( lineOfText : Text )**

| Parameter Type | Description           |
|----------------|-----------------------|
| lineOfText     | Text -> Text to write |

### Description

The `.writeLine()` function writes *lineOfText* content at the current position and inserts an end-of-line delimiter (unlike the [.writeText\(\)](#) function). By default, a native

end-of-line delimiter is used, but you can define another delimiter when [opening the file handle](#) by setting the `.breakModeWrite` property.

When this function is executed, the current position (`.offset`) is updated after the end-of-line delimiter.

## See also

[.breakModeWrite](#), [.readLine\(\)](#), [.writeText\(\)](#)

## .writeText()

- History

**.writeText( *textToWrite* : Text )**

**Parameter Type Description**

*textToWrite* Text -> Text to write

## Description

The `.writeText()` function writes *textToWrite* content at the current position and does not insert a final end-of-line delimiter (unlike the [.writeLine\(\)](#) function). This function replaces all original end-of-line delimiters. By default, the native delimiter is used, but you can define another delimiter when [opening the file handle](#) by setting the `.breakModeWrite` property.

When this function is executed, the current position (`.offset`) is updated after the next end-of-line delimiter.

## See also

[.breakModeWrite](#), [.readText\(\)](#), [.writeLine\(\)](#)

# Folder

`Folder` objects are created with the [Folder](#) command. They contain references to folders that may or may not actually exist on disk. For example, when you execute the `Folder` command to create a new folder, a valid `Folder` object is created but nothing is actually stored on disk until you call the [`folder.create\(\)`](#) function.

## Example

The following example creates a "JohnSmith" folder:

```
Form.curfolder:=Folder(fk database folder)
Form.curfolder:=Folder("C:\\Users\\JohnSmith\\";fk platform path)
```

## Pathnames

`Folder` objects support several pathnames, including [filesystems](#) or [posix](#) syntax. Supported pathnames are detailed in the [Pathnames](#) page.

## Folder object

**.copyTo( destinationFolder : 4D.Folder { ; newName : Text } { ; overwrite : Integer } ) : 4D.Folder** copies the `Folder` object into the specified `destinationFolder`

**.create() : Boolean** creates a folder on disk according to the properties of the `Folder` object

**.createAlias( destinationFolder : 4D.Folder ; aliasName : Text { ; aliasType : Integer } ) : 4D.File** creates an alias (macOS) or a shortcut (Windows)

**.creationDate : Date** the creation date of the folder

**.creationTime : Time** the creation time of the folder

**.delete( { option : Integer } )** deletes the folder

**.exists : Boolean** true if the folder exists on disk

**.extension : Text** returns the extension of the folder name (if any)

**.file( path : Text ) : 4D.File** a `File` object inside the `Folder` object and returns its reference

**.files( { options : Integer } ) : Collection** a collection of `File` objects contained in the folder

**.folder( path : Text ) : 4D.Folder** creates a `Folder` object inside the parent `Folder` object and returns its reference

**.folders( { options : Integer } ) : Collection** returns a collection of `Folder` objects contained in the parent folder

**.fullName : Text** returns the full name of the folder, including its extension (if any)

**.getIcon( { size : Integer } ) : Picture** returns the icon of the folder

**.hidden : Boolean** true if the folder is set as "hidden" at the system level

**.isAlias : Boolean** always **false** for a `Folder` object

**.isFile : Boolean** always **false** for a folder

**.isFolder : Boolean** always **true** for a folder

**.isPackage : Boolean** true if the folder is a package on macOS (and exists on disk)

**.modificationDate : Date** the date of the folder's last modification

**.modificationTime : Time** the time of the folder's last modification

**.name : Text** the name of the folder, without extension (if any)

**.original : 4D.Folder** the same `Folder` object as the folder

**.parent : 4D.Folder** the parent folder object of the folder

**.path : Text** the POSIX path of the folder

**.platformPath : Text** the path of the folder expressed with the current platform syntax

**.moveTo( destinationFolder : 4D.Folder { ; newName : Text } ) : 4D.Folder** moves or renames the `Folder` object (source folder) into the specified `destinationFolder`

**.rename( newName : Text ) : 4D.Folder** renames the folder with the name you passed in `newName` and returns the renamed `Folder` object

## Folder

- History

**Folder** ( *path* : Text { ; *pathType* : Integer }{ ; } ) : 4D.Folder  
**Folder** ( *folderConstant* : Integer { ; } ) : 4D.Folder

| Parameter             | Type      | Description  |
|-----------------------|-----------|--|
| <i>path</i>           | Text      | -> Folder path   |
| <i>folderConstant</i> | Integer   | -> 4D folder constant  |
| <i>pathType</i>       | Integer   | -> <code>fk posix path</code> (default) or <code>fk platform path</code> |
| *                     |           | -> * to return folder of host database                                   |
| Result                | 4D.Folder | <- New folder object   |

## Description

The `Folder` command creates and returns a new object of the `4D.Folder` type. The command accepts two syntaxes:

### `Folder ( path { ; pathType } { ; * } )`

In the *path* parameter, pass a folder path string. You can use a custom string or a filesystem (e.g., "/DATA").

Only absolute pathnames are supported with the `Folder` command.

By default, 4D expects a path expressed with the POSIX syntax. If you work with platform pathnames (Windows or macOS), you must declare it using the *pathType* parameter. The following constants are available:

| Constant         | Value | Comment   |
|------------------|-------|---|
| fk platform path | 1     | Path expressed with a platform-specific syntax (mandatory in case of platform pathname) |
| fk posix path    | 0     | Path expressed with POSIX syntax (default)  |

### `Folder ( folderConstant { ; * } )`

In the *folderConstant* parameter, pass a 4D built-in or system folder, using one of the following constants:

| Constant                   | Value | Comment   |
|----------------------------|-------|---|
| fk applications folder     | 116   |   |
| fk data folder             | 9     | Associated filesystem: "/DATA"  |
| fk database folder         | 4     | Associated filesystem: "/PACKAGE"   |
| fk desktop folder          | 115   |   |
| fk documents folder        | 117   | Document folder of the user   |
| fk home folder             | 118   | Current home folder of the user (usually <code>/Users/&lt;username&gt;/</code> )                                  |
| fk licenses folder         | 1     | Folder containing the machine's 4D license files  |
| fk logs folder             | 7     | Associated filesystem: "/LOGS"  |
| fk mobileApps folder       | 10    |   |
| fk remote database folder  | 3     | 4D database folder created on each 4D remote machine  |
| fk resources folder        | 6     | Associated filesystem: "/RESOURCES"   |
| fk system folder           | 100   |   |
| fk user preferences folder | 0     | 4D folder that stores user preference files within the user home folder   |
| fk web root folder         | 8     | Current Web root folder of the database: if within the package <code>"/PACKAGE/path"</code> , otherwise full path |

If the command is called from a component, pass the optional *parameter* to get the path of the host database. Otherwise, if you omit the parameter, a null object is always returned.

On Windows, in merged clients, the location of built-in folders is modified if the `ShareLocalResourcesOnWindowsClient` [BuildApp key](#) is used.

## 4D.Folder.new()

- History

**4D.Folder.new** (`path : Text { ; pathType : Integer }{ ; }`) : 4D.Folder  
**4D.Folder.new** (`folderConstant : Integer { ; }`) : 4D.Folder

### Description

The `4D.Folder.new()` function creates and returns a new object of the `4D.Folder` type. It is identical to the [Folder](#) command (shortcut).

It is recommended to use the [Folder](#) shortcut command instead of `4D.Folder.new()`.

## .copyTo()

- History

**.copyTo**( `destinationFolder : 4D.Folder { ; newName : Text } { ; overwrite : Integer }` ) : 4D.Folder

| Parameter                      | Type   | Description   |
|--------------------------------|--|---|
| <code>destinationFolder</code> | <code>4D.Folder-&gt;</code> Destination folder |   |
| <code>newName</code>           | <code>Text</code>                              | -> Name for the copy                                      |
| <code>overwrite</code>         | <code>Integer</code>                           | -> <code>fk overwrite</code> to replace existing elements |
| <code>Result</code>            | <code>4D.Folder&lt;-</code>                    | Copied file or folder                                     |

### Description

The `.copyTo()` function copies the `Folder` object into the specified `destinationFolder`.

The `destinationFolder` must exist on disk, otherwise an error is generated.

By default, the folder is copied with the name of the original folder. If you want to rename the copy, pass the new name in the `newName` parameter. The new name must comply with naming rules (e.g., it must not contain characters such as ":", "/", etc.), otherwise an error is returned.

If a folder with the same name already exists in the `destinationFolder`, by default 4D generates an error. You can pass the `fk overwrite` constant in the `overwrite` parameter to ignore and overwrite the existing file

| Constant                  | Value          | Comment                             |
|---------------------------|----------------|-------------------------------------|
| <code>fk overwrite</code> | <code>4</code> | Overwrite existing elements, if any |

### Returned value

The copied `Folder` object.

### Example

You want to copy a Pictures `folder` from the user's Document folder to the Database

folder:

```
var $userImages; $copiedImages : 4D.Folder  
$userImages:=Folder(fk documents folder+"/Pictures/")  
$copiedImages:=$userImages.copyTo(Folder(fk database folder);fk  
overwrite)
```

## .create()

- History

**.create()** : Boolean

| Parameter | Type       | Description  |
|-----------|------------|--|
| Result    | Boolean <- | True if the folder was created successfully, false otherwise |

### Description

The `.create()` function creates a folder on disk according to the properties of the `Folder` object.

If necessary, the function creates the folder hierarchy as described in the [platformPath](#) or [path](#) properties. If the folder already exists on disk, the function does nothing (no error is thrown) and returns false.

### Returned value

- **True** if the folder is created successfully;
- **False** if a folder with the same name already exists or if an error occurred.

### Example 1

Create an empty folder in the database folder:

```
var $created : Boolean  
$created:=Folder("/PACKAGE/SpecialPrefs").create()
```

### Example 2

Creation of the "/Archives2019/January/" folder in the database folder:

```
$newFolder:=Folder("/PACKAGE/Archives2019/January")  
If($newFolder.create())  
    ALERT("The "+$newFolder.name+" folder was created.")  
Else  
    ALERT("Impossible to create a "+$newFolder.name+" folder.")  
End if
```

## .createAlias()

- History

**.createAlias(** *destinationFolder* : 4D.Folder ; *aliasName* : Text { ; *aliasType* : Integer } **)** : 4D.File

| Parameter         | Type      | Description                                  |
|-------------------|-----------|--|
| destinationFolder | 4D.Folder | Destination folder for the alias or shortcut |
| aliasName         | Text      | -> Name of the alias or shortcut             |
| aliasType         | Integer   | -> Type of the alias link                    |
| Result            | 4D.File   | <- Alias or shortcut reference               |

## Description

The `.createAlias()` function creates an alias (macOS) or a shortcut (Windows) to the folder with the specified *aliasName* name in the folder designated by the *destinationFolder* object.

Pass the name of the alias or shortcut to create in the *aliasName* parameter.

By default on macOS, the function creates a standard alias. You can also create a symbolic link by using the *aliasType* parameter. The following constants are available:

| Constant                      | Value | Comment                    |
|-------------------------------|-------|----------------------------|
| <code>fk alias link</code>    | 0     | Alias link (default)       |
| <code>fk symbolic link</code> | 1     | Symbolic link (macOS only) |

On Windows, a shortcut (.lnk file) is always created (the *aliasType* parameter is ignored).

## Returned object

A `4D.File` object with the `isAlias` property set to **true**.

## Example

You want to create an alias to an archive folder in your database folder:

```
$myFolder:=Folder("C:\\\\Documents\\\\Archives\\\\2019\\\\January";fk platform path)
$aliasFile:=$myFolder.createAlias(Folder("/PACKAGE");"Jan2019")
```

## .creationDate

- History

**.creationDate** : Date

## Description

The `.creationDate` property returns the creation date of the folder.

This property is **read-only**.

## .creationTime

- History

**.creationTime** : Time

## Description

The `.creationTime` property returns the creation time of the folder (expressed as a number of seconds beginning at 00:00).

This property is **read-only**.

## **.delete()**

- History

**.delete( { option : Integer } )**

| Parameter | Type      | Description            |
|-----------|-----------|------------------------|
| option    | Integer-> | Folder deletion option |

### **Description**

The `.delete()` function deletes the folder.

By default, for security reasons, if you omit the option parameter, `.delete()` only allows empty folders to be deleted. If you want the command to be able to delete folders that are not empty, you must use the option parameter with one of the following constants:

| Constant             | Value | Comment  |
|----------------------|-------|--|
| Delete only if empty | 0     | Deletes folder only when it is empty             |
| Delete with contents | 1     | Deletes folder along with everything it contains |

When `Delete only if empty` is passed or if you omit the option parameter:

- The folder is only deleted if it is empty; otherwise, the command does nothing and an error -47 is generated.
- If the folder does not exist, the error -120 is generated.

When `Delete with contents` is passed:

- The folder, along with all of its contents, is deleted. **Warning:** Even when this folder and/or its contents are locked or set to read-only, if the current user has suitable access rights, the folder (and contents) is still deleted.
  - If this folder, or any of the files it contains, cannot be deleted, deletion is aborted as soon as the first inaccessible element is detected, and an error(\*) is returned. In this case, the folder may be only partially deleted. When deletion is aborted, you can use the `GET LAST ERROR STACK` command to retrieve the name and path of the offending file.
  - If the folder does not exist, the command does nothing and no error is returned.
- (\*) Windows: -54 (Attempt to open locked file for writing) macOS: -45 (The file is locked or the pathname is not correct)

## **.exists**

- History

**.exists : Boolean**

### **Description**

The `.exists` property returns true if the folder exists on disk, and false otherwise.

This property is **read-only**.

## .extension

- History

**.extension** : Text

### Description

The `.extension` property returns the extension of the folder name (if any). An extension always starts with ".". The property returns an empty string if the folder name does not have an extension.

This property is **read-only**.

## .file()

- History

**.file( path : Text )** : 4D.File

| Parameter | Type    | Description  |
|-----------|---------|--|
| path      | Text    | -> Relative POSIX file pathname                    |
| Result    | 4D.File | <- <code>File</code> object (null if invalid path) |

### Description

The `.file()` function creates a `File` object inside the `Folder` object and returns its reference.

In *path*, pass a relative POSIX path to designate the file to return. The path will be evaluated from the parent folder as root.

### Returned value

A `File` object or null if *path* is invalid.

## Example

```
var $myPDF : 4D.File  
$myPDF:=Folder(fk documents folder).file("Pictures/info.pdf")
```

## .files()

- History

**.files( { options : Integer } )** : Collection

| Parameter | Type       | Description                            |
|-----------|------------|--|
| options   | Integer    | -> File list options                   |
| Result    | Collection | <- Collection of children file objects |

### Description

The `.files()` function returns a collection of `File` objects contained in the folder.

Aliases or symbolic links are not resolved.

By default, if you omit the *options* parameter, only the files at the first level of the folder are returned in the collection, as well as invisible files or folders. You can modify this by passing, in the *options* parameter, one or more of the following constants:

| Constant                         | Value | Comment  |
|----------------------------------|-------|--|
| <code>fk recursive</code>        | 1     | The collection contains files of the specified folder and its subfolders |
| <code>fk ignore invisible</code> | 8     | Invisible files are not listed   |

## Returned value

Collection of `File` objects.

## Example 1

You want to know if there are invisible files in the Database folder:

```
var $all; $noInvisible : Collection
$all:=Folder(fk database folder).files()
$noInvisible:=Folder(fk database folder).files(fk ignore invisible)
If($all.length#$noInvisible.length)
    ALERT("Database folder contains hidden files.")
End if
```

## Example 2

You want to get all files that are not invisible in the Documents folder:

```
var $recursive : Collection
$recursive:=Folder(fk documents folder).files(fk recursive+fk ignore
invisible)
```

## .folder()

- History

**.folder( *path* : Text ) : 4D.Folder**

| Parameter   | Type        | Description  |
|-------------|-------------|--|
| <i>path</i> | Text        | -> Relative POSIX file pathname                      |
| Result      | 4D.Folder<- | Created folder object (null if invalid <i>path</i> ) |

## Description

The `.folder()` function creates a `Folder` object inside the parent `Folder` object and returns its reference.

In *path*, pass a relative POSIX path to designate the folder to return. The path will be evaluated from the parent folder as root.

## Returned value

A `Folder` object or null if *path* is invalid.

## Example

```
var $mypicts : 4D.Folder  
$mypicts:=Folder(fk documents folder).folder("Pictures")
```

## .folders()

- History

**.folders( { options : Integer } ) : Collection**

| Parameter | Type | Description |
|-----------|------|-------------|
|-----------|------|-------------|

options Integer -> Folder list options

Result Collection <- Collection of children folder objects

### Description

The `.folders()` function returns a collection of `Folder` objects contained in the parent folder.

By default, if you omit the `options` parameter, only the folders at the first level of the folder are returned in the collection. You can modify this by passing, in the `options` parameter, one or more of the following constants:

| Constant                         | Value | Comment  |
|----------------------------------|-------|--|
| <code>fk recursive</code>        | 1     | The collection contains folders of the specified folder and its subfolders |
| <code>fk ignore invisible</code> | 8     | Invisible folders are not listed   |

### Returned value

Collection of `Folder` objects.

## Example

You want the collection of all folders and subfolders of the database folder:

```
var $allFolders : Collection  
$allFolders:=Folder("/PACKAGE").folders(fk recursive)
```

## .fullName

- History

**.fullName : Text**

### Description

The `.fullName` property returns the full name of the folder, including its extension (if any).

This property is **read-only**.

## .getIcon()

- History

**.getIcon( { size : Integer } ) : Picture**

| Parameter | Type           | Description                                      |
|-----------|----------------|--|
| size      | Integer->      | Side length for the returned picture<br>(pixels) |
| Result    | Picture <-Icon |  |

## Description

The `.getIcon()` function returns the icon of the folder.

The optional `size` parameter specifies the dimensions in pixels of the returned icon. This value actually represents the length of the side of the square containing the icon. Icons are usually defined in 32x32 pixels ("large icons") or 16x16 pixels ("small icons"). If you pass 0 or omit this parameter, the "large icon" version is returned.

If the folder does not exist on disk, a default blank icon is returned.

## Returned value

Folder icon [picture](#).

## .hidden

- History

**.hidden** : Boolean

## Description

The `.hidden` property returns true if the folder is set as "hidden" at the system level, and false otherwise.

This property is **read-only**.

## .isAlias

- History

**.isAlias** : Boolean

## Description

The `.isAlias` property returns always **false** for a `Folder` object.

This property is **read-only**.

## .isFile

- History

**.isFile** : Boolean

## Description

The `.isFile` property returns always **false** for a folder.

This property is **read-only**.

## .isFolder

- History

**.isFolder** : Boolean

### Description

The `.isFolder` property returns always **true** for a folder.

This property is **read-only**.

## .isPackage

- History

**.isPackage** : Boolean

### Description

The `.isPackage` property returns true if the folder is a package on macOS (and exists on disk). Otherwise, it returns false.

On Windows, `.isPackage` always returns **false**.

This property is **read-only**.

## .modificationDate

- History

**.modificationDate** : Date

### Description

The `.modificationDate` property returns the date of the folder's last modification.

This property is **read-only**.

## .modificationTime

- History

**.modificationTime** : Time

### Description

The `.modificationTime` property returns the time of the folder's last modification (expressed as a number of seconds beginning at 00:00).

This property is **read-only**.

## .moveTo()

- History

**.moveTo(** *destinationFolder* : 4D.Folder { ; *newName* : Text } **)** : 4D.Folder

| Parameter         | Type        | Description                       |
|-------------------|-------------|-----------------------------------|
| destinationFolder | 4D.Folder-> | Destination folder                |
| newName           | Text        | -> Full name for the moved folder |
| Result            | 4D.Folder<- | Moved folder                      |

## Description

The `.moveTo( )` function moves or renames the `Folder` object (source folder) into the specified *destinationFolder*.

The *destinationFolder* must exist on disk, otherwise an error is generated.

By default, the folder retains its name when moved. If you want to rename the moved folder, pass the new full name in the *newName* parameter. The new name must comply with naming rules (e.g., it must not contain characters such as ":", "/", etc.), otherwise an error is returned.

## Returned object

The moved `Folder` object.

## Example

You want to move and rename a folder:

```
var $tomeove; $moved : Object
$docs:=Folder(fk documents folder)
$tomeove:=$docs.folder("Pictures")
$tomeove2:=$tomeove.moveTo($docs.folder("Archives");"Pic_Archives")
```

## .name

- History

### .name : Text

## Description

The `.name` property returns the name of the folder, without extension (if any).

This property is **read-only**.

## .original

- History

### .original : 4D.Folder

## Description

The `.original` property returns the same Folder object as the folder.

This property is **read-only**.

This property is available on folders to allow generic code to process folders or files.

## .parent

- History

**.parent** : 4D.Folder

### Description

The `.parent` property returns the parent folder object of the folder. If the path represents a system path (e.g., "/DATA/"), the system path is returned.

If the folder does not have a parent (root), the null value is returned.

This property is **read-only**.

## .path

- History

**.path** : Text

### Description

The `.path` property returns the POSIX path of the folder. If the path represents a filesystem (e.g., "/DATA/"), the filesystem is returned.

This property is **read-only**.

## .platformPath

- History

**.platformPath** : Text

### Description

The `.platformPath` property returns the path of the folder expressed with the current platform syntax.

This property is **read-only**.

## .rename()

- History

**.rename( newName : Text )** : 4D.Folder

| Parameter | Type      | Description                     |
|-----------|-----------|---------------------------------|
| newName   | Text      | -> New full name for the folder |
| Result    | 4D.Folder | <- Renamed folder               |

### Description

The `.rename()` function renames the folder with the name you passed in `newName` and returns the renamed `Folder` object.

The `newName` parameter must comply with naming rules (e.g., it must not contain characters such as ":", "/", etc.), otherwise an error is returned. If a file with the same name already exists, an error is returned.

## **Returned object**

The renamed `Folder` object.

## **Example**

```
var $toRename : 4D.Folder  
$toRename:=Folder("/RESOURCES/Pictures").rename("Images")
```

# Function

A **4D.Function** object contains a piece of code that can be executed from an object, either using the `()` operator, or using the `apply()` and `call()` functions. 4D proposes three kinds of `Function` objects:

- **native functions**, i.e. built-in functions from various 4D classes such as `collection.sort()` or `file.copyTo()`.
- **user functions**, created in user `classes` using the [Function keyword](#).
- **formula functions**, i.e. functions that can execute any 4D formula.

## Formula objects

The [Formula](#) and [Formula from string](#) commands allow you to create [4D.Function objects](#) to execute any 4D expression or code expressed as text.

Formula objects can be encapsulated in object properties:

```
var $f : 4D.Function  
$f:=New object  
$f.message:=Formula(ALERT("Hello world"))
```

This property is an "object function", i.e. a function which is bound to its parent object. To execute a function stored in an object property, use the `()` operator after the property name, such as:

```
$f.message() //displays "Hello world"
```

Syntax with brackets is also supported:

```
$f["message"]() //displays "Hello world"
```

Note that, even if it does not have parameters (see below), an object function to be executed must be called with `( )` parenthesis. Calling only the object property will return a new reference to the formula (and will not execute it):

```
$o:=$f.message //returns the formula object in $o
```

You can also execute a function using the `apply()` and `call()` functions:

```
$f.message.apply() //displays "Hello world"
```

## Passing parameters

You can pass parameters to your formulas using the [sequential parameter syntax](#) based upon `$1`, `$2...$n`. For example, you can write:

```
var $f : Object  
$f:=New object  
$f.message:=Formula(ALERT("Hello "+$1))  
$f.message("John") //displays "Hello John"
```

Or using the [`.call\(\)`](#) function:

```
var $f : Object  
$f:=Formula($1+" "+$2)  
$text:=$f.call(Null;"Hello";"World") //returns "Hello World"  
$text:=$f.call(Null;"Welcome to";String(Year of(Current date)))  
//returns "Welcome to 2019" (for example)
```

## Parameters to a single method

For more convenience, when the formula is made of a single project method, parameters can be omitted in the formula object initialization. They can just be passed when the formula is called. For example:

```
var $f : 4D.Function

$f:=Formula(myMethod)
//Writing Formula(myMethod($1;$2)) is not necessary
$text:=$f.call(Null;"Hello";"World") //returns "Hello World"
$text:=$f.call() //returns "How are you?"

//myMethod
#DECLARE ($param1 : Text; $param2 : Text)->$return : Text
If(Count parameters=2)
    $return:=$param1+" "+$param2
Else
    $return:="How are you?"
End if
```

Parameters are received within the method, in the order they are specified in the call.

## Summary

[.apply\(\)](#) : any

[.apply\( thisObj : Object { ; formulaParams : Collection } \)](#) : any executes the `formula` object to which it is applied and returns the resulting value

[.call\(\)](#) : any

[.call\( thisObj : Object { ; ...params : any } \)](#) : any executes the `formula` object to which it is applied and returns the resulting value

[.source : Text](#) contains the source expression of the `formula` as text

## Formula

- History

**Formula** ( *formulaExp* : Expression ) : 4D.Function

| Parameter | Type | Description |
|-----------|------|-------------|
|-----------|------|-------------|

formulaExp Expression -> Formula to be returned as object

Result 4D.Function<- Native function encapsulating the formula

## Description

The `Formula` command creates a `4D Function` object based upon the *formulaExp* expression. *formulaExp* can be as simple as a single value or complex, such as a project method with parameters.

Having a formula as an object allows it to be passed as a parameter (calculated attribute) to commands or methods or to be executed from various components without needing to declare them as "shared by components and host database". When called, the formula object is evaluated within the context of the database or component that created it.

The returned formula can be called with:

- [.call\(\)](#) or [.apply\(\)](#) methods, or

- object notation syntax (see [formula object](#)).

```
var $f : 4D.Function
$f:=Formula(1+2)
$o:=New object ("myFormula";$f)

//three different ways to call the formula
$f.call($o) //returns 3
$f.apply($o) //returns 3
$o.myFormula() //returns 3
```

You can pass [parameters](#) to the `Formula`, as seen below in [example 4](#).

You can specify the object on which the formula is executed, as seen in [example 5](#). The properties of the object can then be accessed via the `This` command.

If *formulaExp* uses local variables, their values are copied and stored in the returned formula object when it is created. When executed, the formula uses these copied values rather than the current value of the local variables. Note that using arrays as local variables is not supported.

The object created by `Formula` can be saved, for example, in a database field or in a blob document.

## Example 1

A simple formula:

```
var $f : 4D.Function
$f:=Formula(1+2)

var $o : Object
$o:=New object ("f";$f)

$result:=$o.f() // returns 3
```

## Example 2

A formula using local variables:

```
$value:=10
$o:=New object ("f";Formula($value))
$value:=20

$result:=$o.f() // returns 10
```

## Example 3

A simple formula using parameters:

```
$o:=New object ("f";Formula($1+$2))
$result:=$o.f(10;20) //returns 30
```

## Example 4

A formula using a project method with parameters:

```
$o:=New object ("f";Formula(myMethod))
$result:=$o.f("param1";"param2") // equivalent to
$result:=myMethod("param1";"param2")
```

## Example 5

Using `This`:

```
$o:=New object("fullName";Formula(This.firstName+" "+This.lastName))
$o.firstName:="John"
$o.lastName:="Smith"
$result:=$o.fullName() //returns "John Smith"
```

## Example 6

Calling a formula using object notation:

```
var $feta; $robot : Object
var $calc : 4D.Function
$robot:=New object("name";"Robot";"price";543;"quantity";2)
$feta:=New object("name";"Feta";"price";12.5;"quantity";5)

$calc:=Formula(This.total:=This.price*This.quantity)

//sets the formula to object properties
$feta.calc:=$calc
$robot.calc:=$calc

//call the formula
$feta.calc() // $feta=
{name:Feta,price:12.5,quantity:5,total:62.5,calc:"[object Formula]"}
$robot.calc() // $robot=
{name:Robot,price:543,quantity:2,total:1086,calc:"[object Formula]"}
```

## Formula from string

- History

**Formula from string**( *formulaString* : Text ) : 4D.Function

| Parameter     | Type        | Description                               |
|---------------|-------------|---|
| formulaString | Text        | ->Text formula to be returned as object   |
| Result        | 4D.Function | <-Native object encapsulating the formula |

### Description

The `Formula from string` command creates a 4D.Function object based upon the *formulaString*. *formulaString* can be as simple as a single value or complex, such as a project method with parameters.

This command is similar to `Formula`, except that it handles a text-based formula. In most cases, it is recommended to use the `Formula` command. `Formula from string` should only be used when the original formula was expressed as text (e.g., stored externally in a JSON file). In this context, using syntax with tokens is highly advised.

Because local variable contents can not be accessed by name in compiled mode, they can not be used in *formulaString*. An attempt to access a local variable with `Formula from string` will result in an error (-10737).

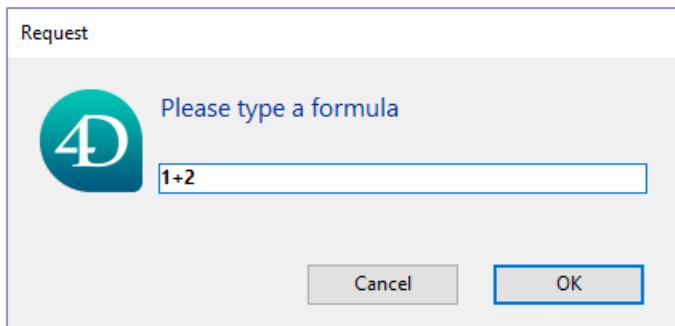
## Example

The following code will create a dialog accepting a formula in text format:

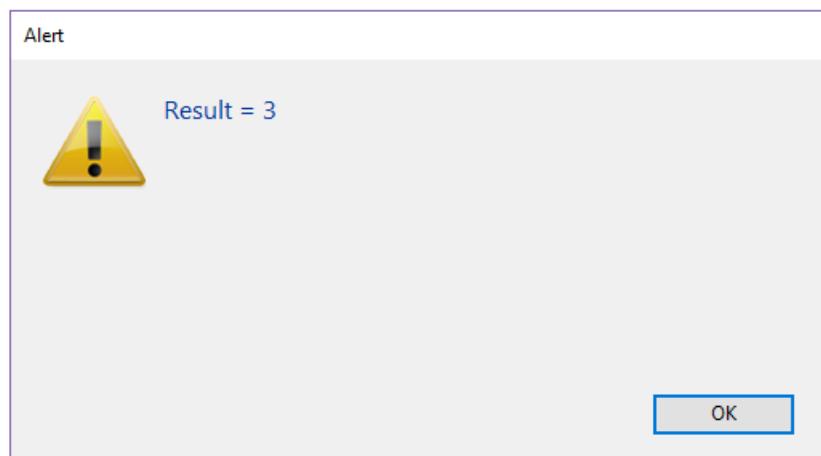
```

var $textFormula : Text
var $f : 4D.Function
$textFormula:=Request("Please type a formula")
If(ok=1)
    $f:=Formula from string($textFormula)
    ALERT("Result = "+String($f.call()))
End if

```



...and execute the formula:



## .apply()

- History

**.apply() : any**

**.apply( thisObj : Object { ; formulaParams : Collection } ) : any**

| Parameter     | Type       | Description   |
|---------------|------------|---|
| thisObj       | Object     | -> Object to be returned by the This command in the formula |
| formulaParams | Collection | values to be passed as \$1...\$n when formula is executed   |
| Result        | any        | <- Value from formula execution                             |

### Description

The `.apply()` function executes the `formula` object to which it is applied and returns the resulting value. The formula object can be created using the `Formula` or `Formula from string` commands.

In the `thisObj` parameter, you can pass a reference to the object to be used as `This` within the formula.

You can also pass a collection to be used as `$1...$n` parameters in the formula using

the optional `formulaParams` parameter.

Note that `.apply()` is similar to `.call()` except that parameters are passed as a collection. This can be useful for passing calculated results.

## Example 1

```
var $f : 4D.Function  
$f:=Formula($1+$2+$3)  
  
$c:=New collection(10;20;30)  
$result:=$f.apply(Null;$c) // returns 60
```

## Example 2

```
var $calc : 4D.Function  
var $feta; $robot : Object  
$robot:=New object("name";"Robot";"price";543;"quantity";2)  
$feta:=New object("name";"Feta";"price";12.5;"quantity";5)  
  
$calc:=Formula(This.total:=This.price*This.quantity)  
  
$calc.apply($feta) // $feta=  
{name:Feta,price:12.5,quantity:5,total:62.5}  
$calc.apply($robot) // $robot=  
{name:Robot,price:543,quantity:2,total:1086}
```

## .call()

- History

`.call() : any`  
`.call( thisObj : Object { ; ...params : any } ) : any`

| Parameter Type | Description   |
|----------------|---|
| thisObj        | Object-> Object to be returned by the <code>This</code> command in the formula  |
| params         | any -> Value(s) to be passed as <code>\$1...\$n</code> when formula is executed |
| Result         | any <- Value from formula execution   |

## Description

The `.call()` function executes the `formula` object to which it is applied and returns the resulting value. The formula object can be created using the `Formula` or `Formula from string` commands.

In the `thisObj` parameter, you can pass a reference to the object to be used as `This` within the formula.

You can also pass values to be used as `$1...$n` parameters in the formula using the optional `params` parameter(s).

Note that `.call()` is similar to `.apply()` except that parameters are passed directly.

## Example 1

```
var $f : 4D.Function  
$f:=Formula(Uppercase($1))  
$result:=$f.call(Null;"hello") // returns "HELLO"
```

## Example 2

```
$o:=New object("value";50)  
$f:=Formula(This.value*2)  
$result:=$f.call($o) // returns 100
```

## .Source

- History

**.source** : Text

## Description

The `.source` property contains the source expression of the `formula` as text.

This property is **read-only**.

## Example

```
var $of : 4D.Function  
var $tf : Text  
$of:=Formula(String(Current time;HH MM AM PM))  
$tf:=$of.source //"String(Current time;HH MM AM PM)"
```

# HTTPRequest

The `HTTPRequest` class allows you to handle [HTTPRequest objects](#) that can be used to configure and send requests to an HTTP server, as well as to process the HTTP server responses.

The `HTTPRequest` class is available from the `4D` class store. You create and send HTTP requests using the [4D.HTTPRequest.new\(\)](#) function, that returns a [HTTPRequest object](#).

- History

## Example

Create a `MyHttpRequestOptions` class for the request options:

```
Class constructor($method : Text; $headers : Object; $body : Text)
This.method:=$method
This.headers:=$headers
This.body:=$body

Function onResponse($request : 4D.HTTPRequest; $event : Object)
//My onResponse method, if you want to handle the request
asynchronously

Function onError($request : 4D.HTTPRequest; $event : Object)
//My onError method, if you want to handle the request asynchronously
```

You can now create your request:

```
var $headers : Object
$headers:=New object()
$headers["field1"] := "value1"

var myHttpRequestOptions : cs.MyHttpRequestOptions
myHttpRequestOptions := cs.MyHttpRequestOptions.new("GET"; $headers;
"")

var $request : 4D.HTTPRequest
$request:=4D.HTTPRequest.new("www.google.com"; myHttpRequestOptions)
$request.wait() //If you want to handle the request synchronously
//Now you can use $request.response to access the result of the request
or $request.error to check the error that happened.
```

## HTTPRequest Object

An `HTTPRequest` object is a non-sharable object.

`HTTPRequest` objects provide the following properties and functions:

**dataType : Text** the `dataType` passed in the `options` object when calling `new()`, "auto" if it was omitted

**encoding : Text** the `encoding` passed in the `options` object when calling `new()`, "UTF-8" if it was omitted

**errors : Collection** the collection of all the errors if at least one error has been triggered

**headers : Object** the `headers` passed in the `options` object when calling `new()`

**method : Text** the `method` passed in the `options` object when calling `new()`

**protocol : Text** the `protocol` passed in the `options` object when calling `new()`

**response : Object** the response to the request if it has received at least the status code, undefined otherwise

**returnResponseBody : Boolean** the `returnResponseBody` passed in the `options` object when calling `new()`

**.terminate()** aborts the HTTP request

**terminated : Boolean** True if the request is terminated (after the call to `onTerminate`), false otherwise

**timeout : Real** the `timeout` passed in the `options` object when calling `new()`

**url : Text** the URL of the HTTP request

**.wait( { time : Real } ) : HTTPRequestClass** waits for the response from the server

## 4D.HTTPRequest.new()

- History

**4D.HTTPRequest.new( url : Text { ; options : Object } ) : 4D.HTTPRequest**

| Parameter | Type           | Description                         |
|-----------|----------------|-------------------------------------|
| url       | Text           | -> URL to which to send the request |
| options   | Object         | -> Request configuration properties |
| Result    | 4D.HTTPRequest | <- New HTTPRequest object           |

### Description

The `4D.HTTPRequest.new()` function creates and sends a HTTP request to the HTTP server defined in `url` with the defined `options`, and returns a `4D.HTTPRequest` object.

The returned `HTTPRequest` object is used to process responses from the HTTP server and call methods.

In `url`, pass the URL where you want to send the request. The syntax to use is:

```
{http://} [{user}]:[{password}]@host[:{port}] [{/path}] [{?queryString}]
{https://} [{user}]:[{password}]@host[:{port}] [{/path}] [{?queryString}]
```

If you omit the protocol part (`http://` or `https://`), a https request is sent.

For example, you can pass the following strings:

```

http://www.myserver.com
www.myserver.com/path
http://www.myserver.com/path?name="jones"
https://www.myserver.com/login
http://123.45.67.89:8083
http://john.smith@123.45.67.89:8083
http://[2001:0db8:0000:0000:0000:ff00:0042:8329]
http://[2001:0db8:0000:0000:0000:ff00:0042:8329]:8080/index.html
(**)

```

## **options parameter**

In the *options* parameter, pass an object that can contain the following properties:

| Property           | Type                     | Description  | Default                  |
|--------------------|--------------------------|--|--------------------------|
| body               | Variant                  | Body of the request (required in case of <code>post</code> or <code>put</code> requests). Can be a text, a blob, or an object. The content-type is determined from the type of this property unless it is set inside the headers                                     | undefined                |
| certificatesFolder | <a href="#">Folder</a>   | Sets the active client certificates folder   | undefined                |
| dataType           | Text                     | Type of the response body attribute.<br>Values: "text", "blob", "object", or "auto". If "auto", the type of the body content will be deduced from its MIME type (object for JSON, text for text, javascript, xml, http message and url encoded form, blob otherwise) | "auto"                   |
| decodeData         | Boolean                  | If true, the data received in the <code>onData</code> callback is uncompressed<br>Used only in case of requests with a <code>body</code> ( <code>post</code> or <code>put</code> methods).   | False                    |
| encoding           | Text                     | Encoding of the request body content if "UTF-8"<br>it's a text, ignored if content-type is set inside the headers  |                          |
| headers            | Object                   | Headers of the request. Syntax:<br><code>headers.key=value</code> ( <code>value</code> can be a Collection if the same key must appear multiple times)   | Empty object             |
| method             | Text                     | "POST", "GET", or other method   | "GET"                    |
| minTLSVersion      | Text                     | Sets the minimum version of TLS:<br>" <code>TLSv1_0</code> ", " <code>TLSv1_1</code> ", " <code>TLSv1_2</code> ", " <code>TLSv1_3</code> "   | " <code>TLSv1_2</code> " |
| onData             | <a href="#">Function</a> | Callback when data from the body is received. It receives two objects as parameters (see below)  | undefined                |
| onError            | <a href="#">Function</a> | Callback when an error occurs. It receives two objects as parameters (see below)   | undefined                |
| onHeaders          | <a href="#">Function</a> | Callback when the headers are received. It receives two objects as parameters (see below)  | undefined                |
| onResponse         | <a href="#">Function</a> | Callback when a response is received. It receives two objects as parameters (see below)  | undefined                |
|                    |                          | Callback when the request is over. It  |                          |

| Property              | Type                                  | Description  | Default   |
|-----------------------|---------------------------------------|--|-----------|
| onTerminate           | Function                              | Callback when the request is over. It receives two objects as parameters (see below)   | undefined |
| protocol              | Text                                  | "auto" or "HTTP1". "auto" means HTTP1 in the current implementation  | "auto"    |
| proxyAuthentication   | <a href="#">authentication object</a> | Object handling proxy authentication   | undefined |
| serverAuthentication  | <a href="#">authentication object</a> | Object handling server authentication  | undefined |
| returnResponseBody    | Boolean                               | If false, the response body is not returned in the <a href="#">response object</a> . Returns an error if false and <code>onData</code> is undefined  | True      |
| timeout               | Real                                  | Timeout in seconds. Undefined = no timeout   | Undefined |
| validateTLCertificate | Boolean                               | If false, 4D does not validate the TLS certificate and does not return an error if it is invalid (i.e. expired, self-signed...). Important: In the current implementation, the Certification Authority itself is not verified. | True      |

## Callback functions

All callback functions receive two object parameters:

| Parameter | Type                               |
|-----------|------------------------------------|
| \$param1  | <a href="#">HTTPRequest object</a> |
| \$param2  | <a href="#">Event object</a>       |

Here is the sequence of callback calls:

1. `onHeaders` is always called once
2. `onData` is called zero or several times (not called if the request does not have a body)
3. If no error occurred, `onResponse` is always called once
4. If an error occurs, `onError` is executed once (and terminates the request)
5. `onTerminate` is always executed once

## event object

An `event` object is returned when a [callback function](#) is called. It contains the following properties:

| Property           | Type              | Description   |
|--------------------|-------------------|---|
| <code>.data</code> | <code>blob</code> | Received data. It is always <i>undefined</i> except in the <code>onData</code> callback |
| <code>.type</code> | <code>text</code> | Type of event. Possible values: "response", "error", "headers", "data", or "terminate"  |

## authentication object

An authentication object handles the `options.serverAuthentication` or

`options.proxyAuthentication` property. It can contain the following properties:

| <b>Property</b> | <b>Type</b> | <b>Description</b>                                       | <b>Default</b> |
|-----------------|-------------|--|----------------|
| name            | Text        | Name used for authentication                             | undefined      |
| passwordText    | Text        | Password used for authentication                         | undefined      |
| method          | Text        | Method used for authentication:"basic", "digest", "auto" | "auto"         |

## .dataType

**dataType** : Text

### Description

The `.dataType` property contains the `dataType` passed in the [options](#) object when calling [new\(\)](#), "auto" if it was omitted.

## .encoding

**encoding** : Text

### Description

The `.encoding` property contains the `encoding` passed in the [options](#) object when calling [new\(\)](#), "UTF-8" if it was omitted.

## .errors

**errors** : Collection

### Description

The `.errors` property contains the collection of all the errors if at least one error has been triggered.

Here is the contents of the `.errors` property:

| <b>Property</b>            | <b>Type</b> | <b>Description</b>   |
|----------------------------|-------------|--|
| errors                     | Collection  | 4D error stack in case of error                              |
| [ ].errCode                | Number      | 4D error code  |
| [ ].message                | Text        | Description of the 4D error                                  |
| [ ].componentSignatureText | Text        | Signature of the internal component which returned the error |

## .headers

**headers** : Object

### Description

The `.headers` property contains the `headers` passed in the [options](#) object when calling [new\(\)](#). If it was omitted, contains an empty object.

## .method

**method** : Text

## Description

The `.method` property contains the `method` passed in the `options` object when calling [new\(\)](#). If it was omitted, contains "GET".

## .protocol

**protocol** : Text

## Description

The `.protocol` property contains the `protocol` passed in the `options` object when calling [new\(\)](#). If it was omitted or if "auto" was used, contains the version of the protocol used.

## .response

- History

**response** : Object

## Description

The `.response` property contains the response to the request if it has received at least the status code, undefined otherwise.

A `response` object is a non-sharable object. It provides the following properties:

| Property                 | Type    | Description   |
|--------------------------|---------|---|
| <code>.body</code>       | Variant | Body of the response. The type of the message is defined according to the <a href="#">dataType</a> property. Undefined if the body has not been received yet  |
| <code>.headers</code>    | Object  | Headers of the response. Header names are returned in lowercase. <code>&lt;headername&gt;.key</code> = value (value can be a collection if the same key appears multiple times). Undefined if the headers have not been received yet.                         |
| <code>.status</code>     | Number  | Status code of the response   |
| <code>.statusText</code> | Text    | Message explaining the status code  |
| <code>.rawHeaders</code> | Object  | Headers of the response. Header names are returned untouched (with their original case). <code>&lt;headerName&gt;.key</code> = value (value can be a collection if the same key appears multiple times). Undefined if the headers have not been received yet. |

## .returnResponseBody

**returnResponseBody** : Boolean

## Description

The `.returnResponseBody` property contains the `returnResponseBody` passed in the `options` object when calling [new\(\)](#). If it was omitted, contains True.

## .terminate()

**terminate()**

| Parameter Type | Description                     |
|----------------|---------------------------------|
|                | Does not require any parameters |

## Description

This function is thread-safe.

The `.terminate()` function aborts the HTTP request. It triggers the `onTerminate` event.

## .terminated

**terminated** : Boolean

## Description

The `.terminated` property contains True if the request is terminated (after the call to `onTerminate`), false otherwise.

## .timeout

**timeout** : Real

## Description

The `.timeout` property contains the `timeout` passed in the [options](#) object when calling [new\(\)](#). If it was omitted, contains Undefined.

## .url

**url** : Text

## Description

The `.url` property contains the URL of the HTTP request.

## .wait()

**.wait( { time : Real } )** : HTTPRequestClass

| Parameter | Type                               | Description   |
|-----------|------------------------------------|---|
| time      | Real                               | -> Maximum time in seconds to wait for the response |
| Result    | 4D.HTTPRequest<-HTTPRequest object |   |

## Description

This function is thread-safe.

The `wait()` function waits for the response from the server.

If a `time` parameter is passed, the function will wait at most the defined number of seconds.

If the response from the server has already arrived, the function returns immediately.



# IMAPTransporter

The `IMAPTransporter` class allows you to retrieve messages from a IMAP email server.

## IMAP Transporter object

IMAP Transporter objects are instantiated with the [IMAP New transporter](#) command. They provide the following properties and functions:

- `.acceptUnsecureConnection : Boolean`** **True** if 4D is allowed to establish an unencrypted connection
- `.addFlags( msgIDs : Collection ; keywords : Object ) : Object`**
- `.addFlags( msgIDs : Text ; keywords : Object ) : Object`**
- `.addFlags( msgIDs : Longint ; keywords : Object ) : Object`** adds flags to the `msgIDs` for the specified `keywords`
- `.append( mailObj : Object ; destinationBox : Text ; options : Object ) : Object`** appends a `mailObj` to the `destinationBox`
- `.authenticationMode : Text`** the authentication mode used to open the session on the mail server
- `.checkConnection() : Object`** checks the connection using information stored in the transporter object
- `.checkConnectionDelay : Integer`** the maximum time (in seconds) allowed prior to checking the connection to the server
- `.connectionTimeOut : Integer`** the maximum wait time (in seconds) allowed to establish a connection to the server
- `.copy( msgsIDs : Collection ; destinationBox : Text ) : Object`**
- `.copy( allMsgs : Integer ; destinationBox : Text ) : Object`** copies the messages defined by `msgsIDs` or `allMsgs` to the `destinationBox` on the IMAP server
- `.createBox( name : Text ) : Object`** creates a mailbox with the given `name`
- `.delete( msgsIDs : Collection ) : Object`**
- `.delete( allMsgs : Integer ) : Object`** sets the "deleted" flag for the messages defined in `msgsIDs` or `allMsgs`
- `.deleteBox( name : Text ) : Object`** permanently removes the mailbox with the given `name` from the IMAP server
- `.expunge() : Object`** removes all messages with the "deleted" flag from the IMAP mail server.
- `.getBoxInfo( { name : Text } ) : Object`** returns a `boxInfo` object corresponding to the current mailbox, or the mailbox `name`
- `.getBoxList( { parameters : Object } ) : Collection`** returns a collection of mailboxes describing all of the available mailboxes
- `.getDelimiter() : Text`** returns the character used to delimit levels of hierarchy in the mailbox name
- `.getMail( msgNumber: Integer { ; options : Object } ) : Object`**
- `.getMail( msgID: Text { ; options : Object } ) : Object`** returns the `Email` object corresponding to the `msgNumber` or `msgID` in the mailbox designated by the `IMAP_transporter`
- `.getMails( ids : Collection { ; options : Object } ) : Object`**
- `.getMails( startMsg : Integer ; endMsg : Integer { ; options : Object } ) : Object`** returns an object containing a collection of `Email` objects
- `.getMIMEAsBlob( msgNumber : Integer { ; updateSeen : Boolean } ) : Blob`**
- `.getMIMEAsBlob( msgID : Text { ; updateSeen : Boolean } ) : Blob`** returns a BLOB containing the MIME contents for the message corresponding to the `msgNumber` or

containing the MIME contents for the message corresponding to the *msgNumber* or *msgID* in the mailbox designated by the `IMAP_transporter`

**.host : Text** the name or the IP address of the host server

**.logFile : Text** the path of the extended log file defined (if any) for the mail connection

**.move( msgsIDs : Collection ; destinationBox : Text ) : Object**

**.move( allMsgs : Integer ; destinationBox : Text ) : Object** moves the messages defined by *msgsIDs* or *allMsgs* to the *destinationBox* on the IMAP server

**.numToID( startMsg : Integer ; endMsg : Integer ) : Collection** converts the sequence numbers to IMAP unique IDs for the messages in the sequential range designated by *startMsg* and *endMsg*

**.removeFlags( msgIDs : Collection ; keywords : Object ) : Object**

**.removeFlags( msgIDs : Text ; keywords : Object ) : Object**

**.removeFlags( msgIDs : Longint ; keywords : Object ) : Object** removes flags from the `msgIDs` for the specified `keywords`

**.renameBox( currentName : Text ; newName : Text ) : Object** changes the name of a mailbox on the IMAP server

**.port : Integer** the port number used for mail transactions

**.searchMails( searchCriteria : Text ) : Collection** searches for messages that match the given *searchCriteria* in the current mailbox

**.selectBox( name : Text { ; state : Integer } ) : Object** selects the *name* mailbox as the current mailbox

**.subscribe( name : Text ) : Object** allows adding or removing of the specified mailbox to/from the IMAP server's set of "subscribed" mailboxes

**.unsubscribe( name : Text ) : Object** removes a mailbox from a set of subscribed mailboxes

**.user : Text** the user name used for authentication on the mail server

## IMAP New transporter

- History

**IMAP New transporter( server : Object ) : 4D.IMAPTransporter**

| Parameter | Type                        | Description                |
|-----------|-----------------------------|----------------------------|
| server    | Object                      | -> Mail server information |
| Result    | 4D.IMAPTransporter<- object |                            |

### Description

The `IMAP New transporter` command configures a new IMAP connection according to the *server* parameter and returns a new *transporter* object. The returned transporter object will then usually be used to receive emails.

In the *server* parameter, pass an object containing the following properties:

| <i>server</i>  | Default value<br>(if omitted)                                       |
|--|---|
| <b>.acceptUnsecureConnection : Boolean</b> True if 4D is allowed to establish an unencrypted connection  | False   |
| <b>.accessTokenOAuth2</b> : Text   |   |
| <b>.accessTokenOAuth2</b> : Object   |   |
| Text string or token object representing OAuth2 authorization credentials. Used only with OAUTH2 <code>authenticationMode</code> . If <code>accessTokenOAuth2</code> is used but <code>authenticationMode</code> is omitted, the OAuth 2 protocol is used (if allowed by the server). Not returned in <a href="#">IMAP transporter</a> object. | none  |
| <b>.authenticationMode</b> : Text    the authentication mode used to open the session on the mail server   | the most secure authentication mode supported by the server is used |
| <b>.checkConnectionDelay</b> : Integer    the maximum time (in seconds) allowed prior to checking the connection to the server   | 300   |
| <b>.connectionTimeOut</b> : Integer    the maximum wait time (in seconds) allowed to establish a connection to the server  | 30  |
| <b>.host</b> : Text    the name or the IP address of the host server   | <i>mandatory</i>  |
| <b>.logFile</b> : Text    the path of the extended log file defined (if any) for the mail connection   | none  |
| <b>.password</b> : Text<br>User password for authentication on the server. Not returned in <a href="#">IMAP transporter</a> object.  | none  |
| <b>.port</b> : Integer    the port number used for mail transactions   | 993   |
| <b>.user</b> : Text    the user name used for authentication on the mail server  | none  |

**Warning:** Make sure the defined timeout is lower than the server timeout, otherwise the client timeout will be useless.

## Result

The function returns an [IMAP transporter object](#). All returned properties are **read-only**.

The IMAP connection is automatically closed when the transporter object is destroyed.

## Example

```
$server:=New object
$server.host:="imap.gmail.com" //Mandatory
$server.port:=993
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXX"
$server.logFile:="LogTest.txt" //log to save in the Logs folder

var $transporter : 4D.IMAPTransporter
$transporter:=IMAP New transporter($server)

$status:=$transporter.checkConnection()
If(Not($status.success))
    ALERT("An error occurred: "+$status.statusText)
End if
```

## 4D.IMAPTransporter.new()

**4D.IMAPTransporter.new( server : Object ) : 4D.IMAPTransporter**

| Parameter | Type                                  | Description                             |
|-----------|---------------------------------------|---|
| server    | Object                                | -> Mail server information              |
| Result    | 4D.IMAPTransporter<-<br><u>object</u> | <a href="#">IMAP transporter object</a> |

### Description

The `4D.IMAPTransporter.new()` function creates and returns a new object of the `4D.IMAPTransporter` type. It is identical to the [IMAP New transporter](#) command (shortcut).

## .acceptUnsecureConnection

- History

**.acceptUnsecureConnection** : Boolean

### Description

The `.acceptUnsecureConnection` property contains **True** if 4D is allowed to establish an unencrypted connection when encrypted connection is not possible.

It contains **False** if unencrypted connections are unallowed, in which case an error is returned when encrypted connection is not possible.

Available secured ports are:

- SMTP
  - 465: SMTPS
  - 587 or 25: SMTP with STARTTLS upgrade if supported by the server.
- IMAP
  - 143: IMAP non-encrypted port
  - 993: IMAP with STARTTLS upgrade if supported by the server
- POP3
  - 110: POP3 non-encrypted port
  - 995: POP3 with STARTTLS upgrade if supported by the server

## .addFlags()

- History

**.addFlags( msgIDs : Collection ; keywords : Object ) : Object**

**.addFlags( msgIDs : Text ; keywords : Object ) : Object**

**.addFlags( msgIDs : Longint ; keywords : Object ) : Object**

| Parameter | Type          | Description  |
|-----------|---------------|--|
| msgIDs    | Collection -> | Collection of strings: Message unique IDs (text)<br>Text: Unique ID of a message<br>Longint (IMAP all): All messages in the selected mailbox |
| keywords  | Object        | -> Keyword flags to add  |
| Result    | Object        | <- Status of the addFlags operation  |

## Description

The `.addFlags()` function adds flags to the `msgIDs` for the specified `keywords`.

In the `msgIDs` parameter, you can pass either:

- a *collection* containing the unique IDs of specific messages or
- the unique ID (*text*) of a single message or
- the following constant (*longint*) for all messages in the selected mailbox:

| Constant Value | Comment                                     |
|----------------|---|
| IMAP all 1     | Select all messages in the selected mailbox |

The `keywords` parameter lets you define the flags to add to `msgIDs`. You can use the following standard flags as well as custom flags (custom flags support depends on the server implementation):

| Property      | Type    | Description                                    |
|---------------|---------|--|
| \$draft       | Boolean | True to add the "draft" flag to the message    |
| \$seen        | Boolean | True to add the "seen" flag to the message     |
| \$flagged     | Boolean | True to add the "flagged" flag to the message  |
| \$answered    | Boolean | True to add the "answered" flag to the message |
| \$deleted     | Boolean | True to add the "deleted" flag to the message  |
| <custom flag> | Boolean | True to add the custom flag to the message     |

The custom flags names must respect this rule: the keyword must be a case-insensitive string excluding control chars and space and can not include any of these characters: `( ) { } % * " \`

- For a keyword to be taken into account it has to be true.
- The interpretation of keyword flags may vary per mail client.

## Returned object

The function returns an object describing the IMAP status:

| Property                   | Type       | Description  |
|----------------------------|------------|--|
| success                    | Boolean    | True if the operation is successful, False otherwise                                     |
| statusText                 | Text       | Status message returned by the IMAP server, or last error returned in the 4D error stack |
| errors                     | Collection | 4D error stack (not returned if a IMAP server response is received)                      |
| [ ].errcode                | Number     | 4D error code  |
| [ ].message                | Text       | Description of the 4D error  |
| [ ].componentSignatureText |            | Signature of the internal component which returned the error                             |

## Example

```

var $options;$transporter;$boxInfo;$status : Object
$options:=New object
$options.host:="imap.gmail.com"
$options.port:=993
$options.user:="4d@gmail.com"
$options.password:="xxxxxx"

// Create transporter
$transporter:=IMAP New transporter($options)

// Select mailbox
$boxInfo:=$transporter.selectBox("INBOX")

// Mark all messages from INBOX as read/seen
$flags:=New object
$flags["$seen"]:=True
$status:=$transporter.addFlags(IMAP all;$flags)

```

## .append()

- History

**.append( *mailObj* : Object ; *destinationBox* : Text ; *options* : Object ) : Object**

| Parameter             | Type                                    | Description |
|-----------------------|---|-------------|
| <i>mailObj</i>        | Object-> Email object                   |             |
| <i>destinationBox</i> | Text -> Mailbox to receive Email object |             |
| <i>options</i>        | Object-> Object containing charset info |             |
| <i>Result</i>         | Object<- Status of the append operation |             |

## Description

The `.append()` function appends a `mailObj` to the `destinationBox`.

In the `mailObj` parameter, pass an Email object. For a comprehensive description of mail properties, see [Email object](#). The `.append()` function supports keyword tags in the Email object's `keywords` attribute.

The optional `destinationBox` parameter lets you pass the name of a mailbox where the `mailObj` will be appended. If omitted, the current mailbox is used.

In the optional `options` parameter, you can pass an object to define the charset and encoding for specific parts of the email. Available properties:

| <b>Property</b>   | <b>Type</b> | <b>Description</b>   |
|-------------------|-------------|--|
| headerCharsetText | Text        | Charset and encoding used for the following parts of the email:<br>subject, attachment filenames, and email name attribute(s).<br>Possible values: See possible charsets table below |
| bodyCharset       | Text        | Charset and encoding used for the html and text body contents of<br>the email. Possible values: See possible charsets table below  |

Possible charsets:

| <b>Constant</b> | <b>Value</b>         | <b>Comment</b>  |
|-----------------|----------------------|---|
| mail mode       | US-ASCII_ISO-2022-JP | <ul style="list-style-type: none"> <li>headerCharset: US-ASCII if possible, Japanese (ISO-2022-JP) &amp; Quoted-printable if possible, otherwise UTF-8 &amp; Quoted-printable</li> <li>bodyCharset: US-ASCII if possible, Japanese (ISO-2022-JP) &amp; 7-bit if possible, otherwise UTF-8 &amp; Quoted-printable</li> </ul> |
| mail mode       | ISO88591             | <ul style="list-style-type: none"> <li>headerCharset: ISO-8859-1 &amp; Quoted-printable</li> <li>bodyCharset: ISO-8859-1 &amp; 8-bit</li> </ul>   |
| mail mode       | US-UTF8              | headerCharset & bodyCharset: US-ASCII if possible, otherwise UTF-8 & Quoted-printable ( <b>default value</b> )  |
| mail mode       | US-UTF8 in base64    | headerCharset & bodyCharset: US-ASCII if possible, otherwise UTF-8 & base64   |

## Returned object

The function returns an object describing the IMAP status:

| <b>Property</b>            | <b>Type</b> | <b>Description</b>   |
|----------------------------|-------------|--|
| success                    | Boolean     | True if the operation is successful, False otherwise                                     |
| statusText                 | Text        | Status message returned by the IMAP server, or last error returned in the 4D error stack |
| errors                     | Collection  | 4D error stack (not returned if a IMAP server response is received)                      |
| [ ].errcode                | Number      | 4D error code  |
| [ ].message                | Text        | Description of the 4D error  |
| [ ].componentSignatureText |             | Signature of the internal component which returned the error                             |

## Example

To save an email in the Drafts mailbox:

```

var $settings; $status; $msg; $imap: Object
$settings:=New object("host"; "domain.com"; "user"; "xxxx"; "password";
"xxxx"; "port"; 993)

$imap:=IMAP New transporter($settings)

$msg:=New object
$msg.from:="xxxx@domain.com"
$msg.subject:="Lorem Ipsum"
$msg.textBody:="Lorem ipsum dolor sit amet, consectetur adipiscing
elit."
$msg.keywords:=New object
$msg.keywords["$seen"]:=True//flag the message as read
$msg.keywords["$draft"]:=True//flag the message as a draft

$status:=$imap.append($msg; "Drafts")

```

## .authenticationMode

- History

**.authenticationMode** : Text

### Description

The `.authenticationMode` property contains the authentication mode used to open the session on the mail server.

By default, the most secured mode supported by the server is used.

Possible values are:

| Value    | Constants                    | Comment                                |
|----------|------------------------------|--|
| CRAM-MD5 | IMAP authentication CRAM MD5 | Authentication using CRAM-MD5 protocol |
| LOGIN    | IMAP authentication login    | Authentication using LOGIN protocol    |
| OAuth2   | IMAP authentication OAuth2   | Authentication using OAuth2 protocol   |
| PLAIN    | IMAP authentication plain    | Authentication using PLAIN protocol    |

## .checkConnection()

- History

**.checkConnection()** : Object

| Parameter | Type               | Description                      |
|-----------|--------------------|----------------------------------|
| Result    | Object<-connection | Status of the transporter object |

### Description

The `.checkConnection()` function checks the connection using information stored in the transporter object, recreates the connection if necessary, and returns the status. This function allows you to verify that the values provided by the user are valid and consistent.

### Returned object

The function sends a request to the mail server and returns an object describing the mail status. This object can contain the following properties:

| Property               | Type       | Description  |
|------------------------|------------|--|
| success                | boolean    | True if the check is successful, False otherwise   |
| status                 | number     | (SMTP only) Status code returned by the mail server (0 in case of an issue unrelated to the mail processing) |
| statusText             | text       | Status message returned by the mail server, or last error returned in the 4D error stack                     |
| errors                 | collection | 4D error stack (not returned if a mail server response is received)  |
| [ ].errCode            | number     | 4D error code  |
| [ ].message            | text       | Description of the 4D error  |
| [ ].componentSignature | text       | Signature of the internal component which returned the error   |

## .checkConnectionDelay

- History

**.checkConnectionDelay** : Integer

### Description

The `.checkConnectionDelay` property contains the maximum time (in seconds) allowed prior to checking the connection to the server. If this time is exceeded between two method calls, the connection to the server will be checked. By default, if the property has not been set in the `server` object, the value is 300.

**Warning:** Make sure the defined timeout is lower than the server timeout, otherwise the client timeout will be useless.

## .connectionTimeOut

- History

**.connectionTimeOut** : Integer

### Description

The `.connectionTimeOut` property contains the maximum wait time (in seconds) allowed to establish a connection to the server. By default, if the property has not been set in the `server` object (used to create the transporter object with `SMTP New transporter`, `POP3 New transporter`, or `IMAP New transporter`), the value is 30.

## .copy()

- History

**.copy(** *msgsIDs* : Collection ; *destinationBox* : Text **)** : Object  
**.copy(** *allMsgs* : Integer ; *destinationBox* : Text **)** : Object

| Parameter      | Type          | Description  |
|----------------|---------------|--|
| msgsIDs        | Collection -> | Collection of message unique IDs (strings)                   |
| allMsgs        | Integer ->    | <code>IMAP all</code> : All messages in the selected mailbox |
| destinationBox | Text ->       | Mailbox to receive copied messages                           |
| Result         | Object <-     | Status of the copy operation                                 |

## Description

The `.copy()` function copies the messages defined by `msgsIDs` or `allMsgs` to the `destinationBox` on the IMAP server.

You can pass:

- in the `msgsIDs` parameter, a collection containing the unique IDs of the specific messages to copy, or
- in the `allMsgs` parameter, the `IMAP all` constant (integer) to copy all messages in the selected mailbox.

The `destinationBox` parameter allows you to pass a text value with the name of the mailbox where the copies of messages will be placed.

## Returned object

The function returns an object describing the IMAP status:

| Property                           | Type       | Description  |
|------------------------------------|------------|--|
| success                            | Boolean    | True if the operation is successful, False otherwise                                     |
| statusText                         | Text       | Status message returned by the IMAP server, or last error returned in the 4D error stack |
| errors                             | Collection | 4D error stack (not returned if a IMAP server response is received)                      |
| <code>[].errcode</code>            | Number     | 4D error code  |
| <code>[].message</code>            | Text       | Description of the 4D error  |
| <code>[].componentSignature</code> | Text       | Signature of the internal component which returned the error                             |

## Example 1

To copy a selection of messages:

```

var $server;$boxInfo;$status : Object
var $mailIds : Collection
var $transporter : 4D.IMAPTransporter

$server:=New object
$server.host:="imap.gmail.com" //Mandatory
$server.port:=993
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXX"

$transporter:=IMAP New transporter($server)

//select mailbox
$boxInfo:=$transporter.selectBox("inbox")

//get collection of message unique IDs
$mailIds:=$transporter.searchMails("subject \"4D new feature:\"")

// copy found messages to the "documents" mailbox
$status:=$transporter.copy($mailIds;"documents")

```

## Example 2

To copy all messages in the current mailbox:

```

var $server;$boxInfo;$status : Object
var $transporter : 4D.IMAPTransporter

$server:=New object
$server.host:="imap.gmail.com" //Mandatory
$server.port:=993
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXX"

$transporter:=IMAP New transporter($server)

//select mailbox

$boxInfo:=$transporter.selectBox("inbox")

// copy all messages to the "documents" mailbox
$status:=$transporter.copy(IMAP all;"documents")

```

## .createBox()

- History

**.createBox( name : Text ) : Object**

| Parameter Type | Description  |
|----------------|--|
| name           | Text -> Name of the new mailbox                    |
| Result         | Object <- Status of the mailbox creation operation |

## Description

The `.createBox()` function creates a mailbox with the given `name`. If the IMAP server's hierarchy separator character appears elsewhere in the mailbox name, the IMAP server will create any parent names needed to create the given mailbox.

In other words, an attempt to create "Projects/IMAP/Doc" on a server in which "/" is the hierarchy separator character will create:

- Only the "Doc" mailbox if "Projects" & "IMAP" already exist.
- "IMAP" & "Doc" mailboxes if only "Projects" already exists.
- "Projects" & "IMAP" & "Doc" mailboxes, if they do not already exist.

In the `name` parameter, pass the name of the new mailbox.

## Returned object

The function returns an object describing the IMAP status:

| Property                   | Type       | Description  |
|----------------------------|------------|--|
| success                    | Boolean    | True if the operation is successful, False otherwise                                     |
| statusText                 | Text       | Status message returned by the IMAP server, or last error returned in the 4D error stack |
| errors                     | Collection | 4D error stack (not returned if a IMAP server response is received)                      |
| [ ].errcode                | Number     | 4D error code  |
| [ ].message                | Text       | Description of the 4D error  |
| [ ].componentSignatureText |            | Signature of the internal component which returned the error                             |

## Example

To create a new "Invoices" mailbox:

```

var $pw : text
var $options; $transporter; $status : object

$options:=New object

$pw:=Request("Please enter your password:")
If (OK=1)
$options.host:="imap.gmail.com"
$options.user:="test@gmail.com"
$options.password:=$pw

$transporter:=IMAP New transporter($options)

$status:=$transporter.createBox("Invoices")

If ($status.success)
ALERT("Mailbox creation successful!")
Else
ALERT("Error: "+$status.statusText)
End if
End if

```

## .delete()

- History

**.delete( msgsIDs : Collection ) : Object**  
**.delete( allMsgs : Integer ) : Object**

| Parameter | Type  | Description                       |
|-----------|---|-----------------------------------|
| msgsIDs   | Collection -> Collection of message unique IDs (strings)                  |                                   |
| allMsgs   | Integer -> <small>IMAP all</small> : All messages in the selected mailbox |                                   |
| Result    | Object  | <- Status of the delete operation |

## Description

The `.delete()` function sets the "deleted" flag for the messages defined in `msgsIDs` or `allMsgs`.

You can pass:

- in the `msgsIDs` parameter, a collection containing the unique IDs of the specific messages to delete, or
- in the `allMsgs` parameter, the `IMAP all` constant (integer) to delete all messages in the selected mailbox.

Executing this function does not actually remove messages. Messages with the "delete" flag can still be found by the [.searchMails\(\)](#) function. Flagged messages are deleted from the IMAP server with the [.expunge\(\)](#) function or by selecting another mailbox or when the [transporter object](#) (created with [IMAP New transporter](#)) is destroyed.

## Returned object

The function returns an object describing the IMAP status:

| Property               | Type       | Description  |
|------------------------|------------|--|
| success                | Boolean    | True if the operation is successful, False otherwise                                     |
| statusText             | Text       | Status message returned by the IMAP server, or last error returned in the 4D error stack |
| errors                 | Collection | 4D error stack (not returned if a IMAP server response is received)                      |
| [ ].errcode            | Number     | 4D error code  |
| [ ].message            | Text       | Description of the 4D error  |
| [ ].componentSignature | Text       | Signature of the internal component which returned the error                             |

## Example 1

To delete a selection of messages:

```

var $server;$boxInfo;$status : Object
var $mailIds : Collection
var $transporter : 4D.IMAPTransporter

$server:=New object
$server.host:="imap.gmail.com" //Mandatory
$server.port:=993
$server.user:="4d@gmail.com"
$server.password:="XXXXXXX"

$transporter:=IMAP New transporter($server)

//select mailbox
$boxInfo:=$transporter.selectBox("Inbox")

//get collection of message unique IDs
$mailIds:=$transporter.searchMails("subject \"Reports\"")

// Delete selected messages
$status:=$transporter.delete($mailIds)

```

## Example 2

To delete all messages in the current mailbox:

```

var $server;$boxInfo;$status : Object
var $transporter : 4D.IMAPTransporter

$server:=New object
$server.host:="imap.gmail.com" //Mandatory
$server.port:=993
$server.user:="4d@gmail.com"
$server.password:="XXXXXXX"

$transporter:=IMAP New transporter($server)

//select mailbox
$boxInfo:=$transporter.selectBox("Junk Email")

// delete all messages in the current mailbox
$status:=$transporter.delete(IMAP all)

```

## .deleteBox()

- History

**.deleteBox( name : Text ) : Object**

| Parameter Type | Description                                       |
|----------------|---|
| name           | Text -> Name of the mailbox to delete             |
| Result         | Object<- Status of the mailbox deletion operation |

### Description

The `.deleteBox()` function permanently removes the mailbox with the given `name` from the IMAP server. Attempting to delete an INBOX or a mailbox that does not exist will generate an error.

In the `name` parameter, pass the name of the mailbox to delete.

- The function cannot delete a mailbox that has child mailboxes if the

- parent mailbox has the "\$Noselect" attribute.
- All messages in the deleted mailbox will also be deleted.
- The ability to delete a mailbox depends on the mail server.

## Returned object

The function returns an object describing the IMAP status:

| Property               | Type       | Description  |
|------------------------|------------|--|
| success                | Boolean    | True if the operation is successful, False otherwise                                     |
| statusText             | Text       | Status message returned by the IMAP server, or last error returned in the 4D error stack |
| errors                 | Collection | 4D error stack (not returned if a IMAP server response is received)                      |
| []].errcode            | Number     | 4D error code  |
| []].message            | Text       | Description of the 4D error  |
| []].componentSignature | Text       | Signature of the internal component which returned the error                             |

## Example

To delete the "Nova Orion Industries" child mailbox from the "Bills" mailbox hierarchy:

```

var $pw; $name : text
var $options; $transporter; $status : object

$options:=New object

$pw:=Request("Please enter your password:")

If (OK=1) $options.host:="imap.gmail.com"
$options.user:="test@gmail.com"
$options.password:=$pw

$transporter:=IMAP New transporter($options)

// delete mailbox
$name:="Bills"+$transporter.getDelimiter()+"Nova Orion Industries"
$status:=$transporter.deleteBox($name)

If ($status.success)
  ALERT("Mailbox deletion successful!")
Else
  ALERT("Error: "+$status.statusText)
End if
End if

```

## .expunge()

- History

**.expunge()** : Object

| Parameter | Type     | Description                     |
|-----------|----------|---------------------------------|
| Result    | Object<- | Status of the expunge operation |

## Description

The `.expunge()` function removes all messages with the "deleted" flag from the IMAP mail server. The "deleted" flag can be set with the `.delete()` or `.addFlags()` methods.

## Returned object

The function returns an object describing the IMAP status:

| Property                            | Type       | Description  |
|-------------------------------------|------------|--|
| success                             | Boolean    | True if the operation is successful, False otherwise                                     |
| statusText                          | Text       | Status message returned by the IMAP server, or last error returned in the 4D error stack |
| errors                              | Collection | 4D error stack (not returned if a IMAP server response is received)                      |
| <code>[] .errcode</code>            | Number     | 4D error code  |
| <code>[] .message</code>            | Text       | Description of the 4D error  |
| <code>[] .componentSignature</code> | Text       | Signature of the internal component which returned the error                             |

## Example

```

var $options;$transporter;$boxInfo;$status : Object
var $ids : Collection

$options:=New object
$options.host:="imap.gmail.com"
$options.port:=993
$options.user:="4d@gmail.com"
$options.password:="xxxxxx"

// Create transporter
$transporter:=IMAP New transporter($options)

// Select mailbox
$boxInfo:=$transporter.selectBox("INBOX")

// Find and delete all seen messages in INBOX
$ids:=$transporter.searchMails("SEEN")
$status:=$transporter.delete($ids)

// Purge all messages flagged as deleted
$status:=$transporter.expunge()

```

## .getBoxInfo()

- History

**.getBoxInfo( { name : Text } ) : Object**

| Parameter | Type                   | Description            |
|-----------|------------------------|------------------------|
| name      | Text                   | -> Name of the mailbox |
| Result    | Object<-boxInfo object |                        |

## Description

The `.getBoxInfo()` function returns a `boxInfo` object corresponding to the current

mailbox, or the mailbox *name*. This function returns the same information as [.selectBox\(\)](#) without changing the current mailbox.

In the optional *name* parameter, pass the name of the mailbox to access. The name represents an unambiguous left-to-right hierarchy with levels separated by a specific delimiter character. The delimiter can be found with the [.getDelimiter\(\)](#) function.

If the mailbox *name* is not selectable or does not exist, the function generates an error and returns **null**.

## Returned object

The `boxInfo` object returned contains the following properties:

| Property   | Type   | Description   |
|------------|--------|---|
| name       | text   | Name of the mailbox   |
| mailCount  | number | Number of messages in the mailbox                                   |
| mailRecent | number | Number of messages with the "recent" flag (indicating new messages) |
| id         | text   | Unique id of the mailbox  |

## Example

```
var $transporter : 4D.IMAPTransporter
$transporter:=IMAP New transporter($server)

$info:=$transporter.getBoxInfo("INBOX")
ALERT("INBOX contains "+String($info.mailRecent)+" recent emails.")
```

## .getBoxList()

- History

**.getBoxList( { parameters : Object } ) : Collection**

| Parameter  | Type       | Description                      |
|------------|------------|----------------------------------|
| parameters | Object     | -> Parameter object              |
| Result     | Collection | <- Collection of mailbox objects |

## Description

The `.getBoxList()` function returns a collection of mailboxes describing all of the available mailboxes. This function allows you to locally manage the list of messages located on the IMAP mail server.

In the optional `parameters` parameter, pass an object containing values to filter the returned mailboxes. You can pass:

| Property     | Type    | Description  |
|--------------|---------|--|
| isSubscribed | Boolean | <ul style="list-style-type: none"><li>• <b>True</b> to return only subscribed mailboxes</li><li>• <b>False</b> to return all available mailboxes</li></ul> |

## Result

Each object of the returned collection contains the following properties:

| <b>Property</b> | <b>Type</b> | <b>Description</b>  |
|-----------------|-------------|---|
| [].name         | text        | Name of the mailbox<br>Indicates whether or not the access rights allow the mailbox to be selected:   |
| [].selectable   | boolean     | <ul style="list-style-type: none"> <li>• true - the mailbox can be selected</li> <li>• false - the mailbox can not be selected</li> </ul>   |
|                 |             | Indicates whether or not the access rights allow creating a lower hierarchy in the mailbox:   |
| [].inferior     | boolean     | <ul style="list-style-type: none"> <li>• true - a lower level can be created</li> <li>• false - a lower level can not be created</li> </ul>   |
|                 |             | Indicates if the mailbox has been marked "interesting" by the server:   |
| [].interesting  | boolean     | <ul style="list-style-type: none"> <li>• true - The mailbox has been marked "interesting" by the server. For example, it may contain new messages.</li> <li>• false - The mailbox has not been marked "interesting" by the server.</li> </ul> |

If the account does not contain any mailboxes, an empty collection is returned.

- If there is no open connection, `.getBoxList()` will open a connection.
- If the connection has not been used since the designated connection delay (see `IMAP New transporter`), the `.checkConnection( )` function is automatically called.

## Example

```
var $transporter : 4D.IMAPTransporter
$transporter:=IMAP New transporter($server)

$boxList:=$transporter.getBoxList()

For each($box;$boxList)
  If($box.interesting)
    $split:=Split string($box.name;$transporter.getDelimiter())
    ALERT("New emails are available in the box:
"+$split[$split.length-1])
  End if
End for each
```

## .getDelimiter()

- History

### .getDelimiter() : Text

| <b>Parameter</b> | <b>Type</b> | <b>Description</b>              |
|------------------|-------------|---------------------------------|
| Result           | Text        | <-Hierarchy delimiter character |

## Description

The `.getDelimiter()` function returns the character used to delimit levels of hierarchy in the mailbox name.

The delimiter is a character which can be used to:

- create lower level (inferior) mailboxes
- search higher or lower within the mailbox hierarchy

## Result

Mailbox name delimiter character.

- If there is no open connection, `.getDelimiter()` will open a connection.
- If the connection has not been used since the [designated connection delay](#), the `.checkConnection()` function is automatically called.

## Example

```
var $transporter : 4D.IMAPTransporter
$transporter:=IMAP New transporter($server)

$boxList:=$transporter.getBoxList()

For each($box;$boxList)
  If($box.interesting)
    $split:=Split string($box.name;$transporter.getDelimiter())
    ALERT("New emails are available in the box:
"+$split[$split.length-1])
  End if
End for each
```

## .getMail()

- History

`.getMail( msgNumber: Integer { ; options : Object } ) : Object`  
`.getMail( msgID: Text { ; options : Object } ) : Object`

| Parameter | Type      | Description                      |
|-----------|-----------|----------------------------------|
| msgNumber | Integer-> | Sequence number of the message   |
| msgID     | Text      | -> Unique ID of the message      |
| options   | Object    | -> Message handling instructions |
| Result    | Object    | <- <a href="#">Email object</a>  |

## Description

The `.getMail()` function returns the [Email](#) object corresponding to the *msgNumber* or *msgID* in the mailbox designated by the [IMAP\\_transporter](#). This function allows you to locally handle the email contents.

In the first parameter, you can pass either:

- *msgNumber*, an *integer* value indicating the sequence number of the message to retrieve or
- *msgID*, a *text* value indicating the unique ID of the message to retrieve.

The optional *options* parameter allows you pass an object defining additional instructions for handling the message. The following properties are available:

| <b>Property</b> | <b>Type</b> | <b>Description</b>  |
|-----------------|-------------|---|
| updateSeen      | boolean     | If True, the message is marked as "seen" in the mailbox. If False, the message is not marked as "seen". Default value: True |
| withBody        | boolean     | Pass True to return the body of the message. If False, only the message header is returned. Default value: True             |

- The function generates an error and returns **Null** if *msgID* designates a non-existing message,
  - If no mailbox is selected with the `.selectBox()` function, an error is generated,
  - If there is no open connection, `.getMail()` will open a connection the last mailbox specified with `.selectBox()`.

## Result

`.getMail()` returns an [Email object](#) with the following specific IMAP properties: `id`, `receivedAt`, and `size`.

## Example

You want to get the message with ID = 1:

```
var $server : Object
var $info; $mail; $boxInfo : Variant
var $transporter : 4D.IMAPTransporter

$server:=New object
$server.host:="imap.gmail.com" //Mandatory
$server.port:=993
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXXX"

//create transporter
$transporter:=IMAP New transporter($server)

//select mailbox
$boxInfo:=$transporter.selectBox("Inbox")

//get Email object with ID 1
$mail:=Stransporter.getMail(1)
```

`.getMails()`

- #### ■ History

```
.getMails( ids : Collection { ; options : Object } ) : Object  
.getMails( startMsg : Integer ; endMsg : Integer { ; options : Object } ) : Object
```

| Parameter | Type                                   | Description   |
|-----------|--|---|
| ids       | Collection -> Collection of message ID |   |
| startMsg  | Integer                                | -> Sequence number of the first message   |
| endMsg    | Integer                                | -> Sequence number of the last message  |
| options   | Object                                 | -> Message handling instructions<br>Object containing:  |
| Result    | Object                                 | <- • a collection of <a href="#">Email objects</a> and<br>• a collection of IDs or numbers for missing messages, if any |

## Description

The `.getMails()` function returns an object containing a collection of `Email` objects.

### First Syntax:

**`.getMails( ids { ; options } ) -> result`**

The first syntax allows you to retrieve messages based on their IDs.

In the `ids` parameter, pass a collection of IDs for the messages to return. You can get the IDs with [.getMail\(\)](#).

The optional `options` parameter allows you to define the parts of the messages to be returned. See the **Options** table below for a description of the available properties.

### Second syntax:

**`.getMails( startMsg ; endMsg { ; options } ) -> result`**

The second syntax allows you to retrieve messages based on a sequential range. The values passed represent the position of the messages in the mailbox.

In the `startMsg` parameter, pass an *integer* value corresponding to the number of the first message in a sequential range. If you pass a negative number (`startMsg <= 0`), the first message of the mailbox will be used as the beginning of the sequence.

In the `endMsg` parameter, pass an *integer* value corresponding to the number of the last message to be included in a sequential range. If you pass a negative number (`endMsg <= 0`), the last message of the mailbox will be used as the end of the sequence.

The optional `options` parameter allows you to define the parts of the messages to be returned.

## Options

| Property | Type    | Description   |
|----------|---------|---|
|          |         | If True, the specified messages are marked as "seen" in the updateSeen Boolean mailbox. If False, the messages are not marked as "seen". Default value: True  |
| withBody | Boolean | Pass True to return the body of the specified messages. If False, only the message headers are returned. Default value: True <ul style="list-style-type: none"><li>• If no mailbox is selected with the <a href="#">.selectBox()</a> command, an error is generated.</li><li>• If there is no open connection, <code>.getMails()</code> will open a connection to the last mailbox specified with <a href="#">.selectBox()</a>.</li></ul> |

## Result

`.getMails()` returns an object containing the following collections:

| Property           | Type       | Description  |
|--------------------|------------|--|
| list               | Collection | Collection of <a href="#">Email objects</a> . If no Email objects are found, an empty collection is returned.  |
| notFoundCollection | Collection | <ul style="list-style-type: none"> <li>first syntax - previously passed message IDs that do not exist</li> <li>second syntax - sequence numbers of messages between startMsg and endMsg that do not exist</li> </ul> |
|                    |            | An empty collection is returned if all messages are found.   |

## Example

You want to retrieve the 20 most recent emails without changing their "seen" status:

```

var $server,$boxInfo,$result : Object
var $transporter : 4D.IMAPtransporter

$server:=New object
$server.host:="imap.gmail.com" //Mandatory
$server.port:=993
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXX"

//create transporter
$transporter:=IMAP New transporter($server)

//select mailbox
$boxInfo:=$transporter.selectBox("INBOX")

If ($boxInfo.mailCount>0)
  // retrieve the headers of the last 20 messages without marking them
as read
  $result:=$transporter.getMails($boxInfo.mailCount-
20;$boxInfo.mailCount;\ 
    New object("withBody";False;"updateSeen";False))
    For each($mail;$result.list)
      // ...
    End for each
End if

```

## .getMIMEAsBlob()

- History

**.getMIMEAsBlob( msgNumber : Integer { ; updateSeen : Boolean } ) : Blob**  
**.getMIMEAsBlob( msgID : Text { ; updateSeen : Boolean } ) : Blob**

| Parameter  | Type    | Description  |
|------------|---------|--|
| msgNumber  | Integer | -> Sequence number of the message  |
| msgID      | Text    | -> Unique ID of the message  |
| updateSeen | Boolean | -> If True, the message is marked "seen" in the mailbox. If False the message is left untouched. |
| Result     | BLOB    | <- Blob of the MIME string returned from the mail server   |

## Description

The `.getMIMEAsBlob()` function returns a BLOB containing the MIME contents for the message corresponding to the `msgNumber` or `msgID` in the mailbox designated by the `IMAP_transporter`.

In the first parameter, you can pass either:

- *msgNumber*, an *integer* value indicating the sequence number of the message to retrieve or
- *msgID*, a *text* value indicating the unique ID of the message to retrieve.

The optional *updateSeen* parameter allows you to specify if the message is marked as "seen" in the mailbox. You can pass:

- **True** - to mark the message as "seen" (indicating the message has been read)
- **False** - to leave the message's "seen" status untouched
  - The function returns an empty BLOB if *msgNumber* or *msgID*\* designates a non-existing message,
  - If no mailbox is selected with the [.selectBox\(\)](#) command, an error is generated,
  - If there is no open connection, [.getMIMEAsBlob\(\)](#) will open a connection the last mailbox specified with [.selectBox\(\)](#).

## Result

[.getMIMEAsBlob\(\)](#) returns a [BLOB](#) which can be archived in a database or converted to an [Email object](#) with the [MAIL Convert from MIME](#) command.

## Example

```
var $server : Object
var $boxInfo : Variant
var $blob : Blob
var $transporter : 4D.IMAPtransporter

$server:=New object
$server.host:="imap.gmail.com"
$server.port:=993
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXX"

//create transporter
$transporter:=IMAP New transporter($server)

//select mailbox
$boxInfo:=$transporter.selectBox("Inbox")

//get BLOB
$blob:=$transporter.getMIMEAsBlob(1)
```

## .host

- History

**.host** : Text

## Description

The [.host](#) property contains the name or the IP address of the host server. Used for mail transactions (SMTP, POP3, IMAP).

## .logFile

- History

## .logFile : Text

### Description

The `.logFile` property contains the path of the extended log file defined (if any) for the mail connection. It can be relative (to the current Logs folder) or absolute.

Unlike regular log files (enabled via the `SET DATABASE PARAMETER` command), extended log files store MIME contents of all sent mails and do not have any size limit. For more information about extended log files, refer to:

- **SMTP connections** - [4DSMTPLLog.txt](#)
- **POP3 connections** - [4DPOP3Log.txt](#)
- **IMAP connections** - [4DIMAPLog.txt](#)

## .move()

### ▪ History

**.move(** *msgsIDs* : Collection ; *destinationBox* : Text **)** : Object  
**.move(** *allMsgs* : Integer ; *destinationBox* : Text **)** : Object

| Parameter             | Type  | Description |
|-----------------------|---|-------------|
| <i>msgsIDs</i>        | Collection -> Collection of message unique IDs (strings)                |             |
| <i>allMsgs</i>        | Integer -> <code>IMAP all</code> : All messages in the selected mailbox |             |
| <i>destinationBox</i> | Text -> Mailbox to receive moved messages                               |             |
| <i>Result</i>         | Object <- Status of the move operation                                  |             |

### Description

The `.move()` function moves the messages defined by *msgsIDs* or *allMsgs* to the *destinationBox* on the IMAP server.

You can pass:

- in the *msgsIDs* parameter, a collection containing the unique IDs of the specific messages to move, or
- in the *allMsgs* parameter, the `IMAP all` constant (integer) to move all messages in the selected mailbox.

The *destinationBox* parameter allows you to pass a text value with the name of the mailbox where the messages will be moved.

This function is only supported by IMAP servers compliant with RFC [8474](#).

### Returned object

The function returns an object describing the IMAP status:

| <b>Property</b>            | <b>Type</b> | <b>Description</b>   |
|----------------------------|-------------|--|
| success                    | Boolean     | True if the operation is successful, False otherwise                                     |
| statusText                 | Text        | Status message returned by the IMAP server, or last error returned in the 4D error stack |
| errors                     | Collection  | 4D error stack (not returned if a IMAP server response is received)                      |
| [ ].errcode                | Number      | 4D error code  |
| [ ].message                | Text        | Description of the 4D error  |
| [ ].componentSignatureText |             | Signature of the internal component which returned the error                             |

## Example 1

To move a selection of messages:

```

var $server;$boxInfo;$status : Object
var $mailIds : Collection
var $transporter : 4D.IMAPTransporter

$server:=New object
$server.host:="imap.gmail.com" //Mandatory
$server.port:=993
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXX"

$transporter:=IMAP New transporter($server)

//select mailbox
$boxInfo:=$transporter.selectBox("inbox")

//get collection of message unique IDs
$mailIds:=$transporter.searchMails("subject \"4D new feature:\"")

// Move found messages from the current mailbox to the "documents" mailbox
$status:=$transporter.move($mailIds;"documents")

```

## Example 2

To move all messages in the current mailbox:

```

var $server;$boxInfo;$status : Object
var $transporter : 4D.IMAPTransporter

$server:=New object
$server.host:="imap.gmail.com" //Mandatory
$server.port:=993
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXX"

$transporter:=IMAP New transporter($server)

//select mailbox
$boxInfo:=$transporter.selectBox("inbox")

// move all messages in the current mailbox to the "documents" mailbox
$status:=$transporter.move(IMAP all;"documents")

```

## .numToID()

- History

**.numToID( startMsg : Integer ; endMsg : Integer ) : Collection**

| Parameter | Type       | Description                             |
|-----------|------------|---|
| startMsg  | Integer    | -> Sequence number of the first message |
| endMsg    | Integer    | -> Sequence number of the last message  |
| Result    | Collection | <- Collection of unique IDs             |

### Description

The `.numToID()` function converts the sequence numbers to IMAP unique IDs for the messages in the sequential range designated by *startMsg* and *endMsg* in the currently selected mailbox.

In the *startMsg* parameter, pass an integer value corresponding to the number of the first message in a sequential range. If you pass a negative number (*startMsg* <= 0), the first message of the mailbox will be used as the beginning of the sequence.

In the *endMsg* parameter, pass an integer value corresponding to the number of the last message to be included in a sequential range. If you pass a negative number (*endMsg* <= 0), the last message of the mailbox will be used as the end of the sequence.

### Result

The function returns a collection of strings (unique IDs).

### Example

```
var $transporter : 4D.IMAPTransporter
var $server;$boxInfo;$status : Object
var $mailIds : Collection

$server:=New object
$server.host:="imap.gmail.com" //Mandatory
$server.port:=993
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXX"

$transporter:=IMAP New transporter($server)

//select mailbox
$boxInfo:=$transporter.selectBox("inbox")

//get IDs for 5 last messages received
$mailIds:=$transporter.numToID(( $boxInfo.mailCount-
5);$boxInfo.mailCount)

//delete the messages from the current mailbox
$status:=$transporter.delete($mailIds)
```

## .removeFlags()

- History

**.removeFlags( msgIDs : Collection ; keywords : Object ) : Object**

**.removeFlags(** *msgIDs* : Text ; *keywords* : Object ) : Object  
**.removeFlags(** *msgIDs* : Longint ; *keywords* : Object ) : Object

| Parameter       | Type          | Description  |
|-----------------|---------------|--|
|                 |               | Collection of strings: Message unique IDs (text)   |
| <i>msgIDs</i>   | Collection -> | Text: Unique ID of a message<br>Longint (IMAP all): All messages in the selected mailbox |
| <i>keywords</i> | Object        | -> Keyword flags to remove   |
| <i>Result</i>   | Object        | <- Status of the removeFlags operation   |

## Description

The `.removeFlags()` function removes flags from the `msgIDs` for the specified `keywords`.

In the `msgIDs` parameter, you can pass either:

- a *collection* containing the unique IDs of specific messages or
- the unique ID (*text*) of a single message or
- the following constant (*longint*) for all messages in the selected mailbox:

| Constant | Value | Comment                                     |
|----------|-------|---|
| IMAP all | 1     | Select all messages in the selected mailbox |

The `keywords` parameter lets you define the flags to remove from `msgIDs`. You can use the following standard flags as well as custom flags:

| Parameter     | Type    | Description   |
|---------------|---------|---|
| \$draft       | Boolean | True to remove the "draft" flag from the message    |
| \$seen        | Boolean | True to remove the "seen" flag from the message     |
| \$flagged     | Boolean | True to remove the "flagged" flag from the message  |
| \$answered    | Boolean | True to remove the "answered" flag from the message |
| \$deleted     | Boolean | True to remove the "deleted" flag from the message  |
| <custom flag> | Boolean | True to remove the custom flag from the message     |

Please refer to [.addFlags\(\)](#) for more information on custom flags.

- For a keyword to be taken into account it has to be true.

## Returned object

The function returns an object describing the IMAP status:

| <b>Property</b>            | <b>Type</b> | <b>Description</b>   |
|----------------------------|-------------|--|
| success                    | Boolean     | True if the operation is successful, False otherwise                                     |
| statusText                 | Text        | Status message returned by the IMAP server, or last error returned in the 4D error stack |
| errors                     | Collection  | 4D error stack (not returned if a IMAP server response is received)                      |
| [ ].errcode                | Number      | 4D error code  |
| [ ].message                | Text        | Description of the 4D error  |
| [ ].componentSignatureText |             | Signature of the internal component which returned the error                             |

## Example

```

var $options;$transporter;$boxInfo;$status : Object
$options:=New object
$options.host:="imap.gmail.com"
$options.port:=993
$options.user:="4d@gmail.com"
$options.password:="xxxxxx"

// Create transporter
$transporter:=IMAP New transporter($options)

// Select mailbox
$boxInfo:=$transporter.selectBox("INBOX")

// Mark all messages from INBOX as unseen
$flags:=New object
$flags["$seen"]:=True
$status:=$transporter.removeFlags(IMAP all;$flags)

```

## .renameBox()

- History

**.renameBox( *currentName* : Text ; *newName* : Text ) : Object**

| <b>Parameter</b> | <b>Type</b> | <b>Description</b>               |
|------------------|-------------|----------------------------------|
| currentName      | Text        | -> Name of the current mailbox   |
| newName          | Text        | -> New mailbox name              |
| Result           | Object<-    | Status of the renaming operation |

## Description

The `.renameBox()` function changes the name of a mailbox on the IMAP server. Attempting to rename a mailbox from a mailbox name that does not exist or to a mailbox name that already exists will generate an error.

In the `currentName` parameter, pass the name of the mailbox to be renamed.

Pass the new name for the mailbox in the `newName` parameter.

## Returned object

The function returns an object describing the IMAP status:

| <b>Property</b>            | <b>Type</b> | <b>Description</b>   |
|----------------------------|-------------|--|
| success                    | Boolean     | True if the operation is successful, False otherwise                                     |
| statusText                 | Text        | Status message returned by the IMAP server, or last error returned in the 4D error stack |
| errors                     | Collection  | 4D error stack (not returned if a IMAP server response is received)                      |
| [ ].errcode                | Number      | 4D error code  |
| [ ].message                | Text        | Description of the 4D error  |
| [ ].componentSignatureText |             | Signature of the internal component which returned the error                             |

## Example

To rename your “Invoices” mailbox to “Bills”:

```

var $pw : text
var $options; $transporter; $status : object

$options:=New object

$pw:=Request("Please enter your password:")

If (OK=1) $options.host:="imap.gmail.com"

$options.user:="test@gmail.com"
$options.password:=$pw

$transporter:=IMAP New transporter($options)

// rename mailbox
$status:=$transporter.renameBox("Invoices"; "Bills")

If ($status.success)
  ALERT("Mailbox renaming successful!")
Else
  ALERT("Error: "+$status.statusText)
End if
End if

```

## .port

- History

**.port** : Integer

## Description

The **.port** property contains the port number used for mail transactions. By default, if the **port** property has not been set in the **server** object (used to create the transporter object with **SMTP New transporter**, **POP3 New transporter**, **IMAP New transporter**), the port used is:

- **SMTP** - 587
- **POP3** - 995
- **IMAP** - 993

## .searchMails()

- History

.searchMails( *searchCriteria* : Text ) : Collection

| Parameter      | Type          | Description                   |
|----------------|---------------|-------------------------------|
| searchCriteria | Text          | -> Search criteria            |
| Result         | Collection <- | Collection of message numbers |

## Description

This function is based upon the specification for the [IMAP protocol](#).

The `.searchMails()` function searches for messages that match the given *searchCriteria* in the current mailbox. *searchCriteria* consists of one or more search keys.

*searchCriteria* is a text parameter listing one or more search keys (see [Authorized search-keys](#) below) associated or not with values to look for. A search key may be a single or multiple items. For example:

```
SearchKey1 = FLAGGED  
SearchKey2 = NOT FLAGGED  
SearchKey3 = FLAGGED DRAFT
```

Matching is usually not case-sensitive

- If the *searchCriteria* is a null string, the search will be equivalent to a “select all”.
- If the *searchCriteria* includes multiple search keys, the result is the intersection (AND function) of all the messages that match those keys.

```
searchCriteria = FLAGGED FROM "SMITH"
```

... returns all messages with `¥Flagged` flag set AND sent by Smith.

- You can use the **OR** or **NOT** operators as follows:

```
searchCriteria = OR SEEN FLAGGED
```

... returns all messages with `¥Seen` flag set OR `¥Flagged` flag set

```
searchCriteria = NOT SEEN
```

... returns all messages with `¥Seen` flag not set.

```
searchCriteria = HEADER CONTENT-TYPE "MIXED" NOT HEADER CONTENT-TYPE  
"TEXT"...
```

... returns message whose content-type header contains “Mixed” and does not contain “Text”.

```
searchCriteria = HEADER CONTENT-TYPE "E" NOT SUBJECT "o" NOT HEADER  
CONTENT-TYPE "MIXED"
```

... returns message whose content-type header contains “e” and whose Subject header does not contain “o” and whose content-type header is not “ Mixed ”.

As concerns the last two examples, notice that the result of the search is different when you remove the parentheses of the first search key list.

- The *searchCriteria* may include the optional [CHARSET] specification. This consists of the “CHARSET” word followed by a registered [CHARSET] (US ASCII, ISO-

8859). It indicates the charset of the `searchCriteria` string. Therefore, you must convert the `searchCriteria` string into the specified charset if you use the [CHARSET] specification (see the `CONVERT FROM TEXT` or `Convert to text` commands). By default, 4D encodes in Quotable Printable the `searchCriteria` string if it contains extended characters.

```
searchCriteria = CHARSET "ISO-8859" BODY "Help"
```

... means the search criteria uses the charset iso-8859 and the server will have to convert the search criteria before searching, if necessary.

## Search value types

Search-keys may request the value to search for:

- **Search-keys with a date value:** the date is a string that must be formatted as follows: `date-day+ "-" +date-month+ "-" +date-year` where date-day indicates the number of the day of the month (max. 2 characters), date-month indicates the name of the month (Jan/Feb/Mar/Apr/May/Jun/Jul/Aug/Sep/Oct/Dec) and date-year indicates the year (4 characters). Example: `searchCriteria = SENTBEFORE 1-Feb-2020` (a date does not usually need to be quoted since it does not contain any special characters)
- **Search-keys with a string value:** the string may contain any character and must be quoted. If the string does not contain any special characters, like the space character for instance, it does not need to be quoted. Quoting such strings will ensure that your string value will be correctly interpreted. Example:  
`searchCriteria = FROM "SMITH"` For all search keys that use strings, a message matches the key if the string is a substring of the field. Matching is not case-sensitive.
- **Search-keys with a field-name value:** the field-name is the name of a header field. Example: `searchCriteria = HEADER CONTENT-TYPE "MIXED"`
- **Search-keys with a flag value:** the flag may accept one or several keywords (including standard flags), separated by spaces. Example: `searchCriteria = KEYWORD \Flagged \Draft`
- **Search-keys with a message set value:** Identifies a set of messages. For message sequence numbers, these are consecutive numbers from 1 to the total number of messages in the mailbox. A comma delimits individual numbers; a colon delimits between two numbers inclusive. Examples: `2,4:7,9,12:*` is `2,4,5,6,7,9,12,13,14,15` for a mailbox with 15 messages. `searchCriteria = 1:5 ANSWERED` search in message selection from message sequence number 1 to 5 for messages which have the ¥Answered flag set. `searchCriteria= 2,4 ANSWERED` search in the message selection (message numbers 2 and 4) for messages which have the ¥Answered flag set.

## Authorized search-keys

**ALL:** All messages in the mailbox.

**ANSWERED:** Messages with the ¥Answered flag set.

**UNANSWERED:** Messages that do not have the ¥Answered flag set.

**DELETED:** Messages with the ¥Deleted flag set.

**UNDELETED:** Messages that do not have the ¥Deleted flag set.

**DRAFT:** Messages with the ¥Draft flag set.

**UNDRAFT:** Messages that do not have the ¥Draft flag set.

**FLAGGED**: Messages with the ¥Flagged flag set.

**UNFLAGGED**: Messages that do not have the ¥Flagged flag set.

**RECENT**: Messages that have the ¥Recent flag set.

**OLD**: Messages that do not have the ¥Recent flag set.

**SEEN**: Messages that have the ¥Seen flag set.

**UNSEEN**: Messages that do not have the ¥Seen flag set.

**NEW**: Messages that have the ¥Recent flag set but not the ¥Seen flag. This is functionally equivalent to "(RECENT UNSEEN)".

**KEYWORD flag**: Messages with the specified keyword set.

**UNKEYWORD flag**: Messages that do not have the specified keyword set.

**BEFORE date**: Messages whose internal date is earlier than the specified date.

**ON date**: Messages whose internal date is within the specified date.

**SINCE date**: Messages whose internal date is within or later than the specified date.

**SENTBEFORE date**: Messages whose Date header is earlier than the specified date.

**SENTON date**: Messages whose Date header is within the specified date.

**SENTSINCE date**: Messages whose Date header is within or later than the specified date.

**TO string**: Messages that contain the specified string in the TO header.

**FROM string**: Messages that contain the specified string in the FROM header.

**CC string**: Messages that contain the specified string in the CC header.

**BCC string**: Messages that contain the specified string in the BCC header.

**SUBJECT string**: Messages that contain the specified string in the Subject header.

**BODY string**: Messages that contain the specified string in the message body.

**TEXT string**: Messages that contain the specified string in the header or in the message body.

**HEADER field-name string**: Messages that have a header with the specified field-name and that contain the specified string in the field-body.

**UID message-UID**: Messages with unique identifiers corresponding to the specified unique identifier set.

**LARGER n**: Messages with a size larger than the specified number of bytes.

**SMALLER n**: Messages with a size smaller than the specified number of bytes.

**NOT search-key**: Messages that do not match the specified search key.

**OR search-key1 search-key2**: Messages that match either search key.

## .selectBox()

- History

**.selectBox( name : Text { ; state : Integer } ) : Object**

| Parameter | Type      | Description            |
|-----------|-----------|------------------------|
| name      | Text      | -> Name of the mailbox |
| state     | Integer-> | Mailbox access status  |
| Result    | Object    | <- boxInfo object      |

### Description

The `.selectBox()` function selects the *name* mailbox as the current mailbox. This function allows you to retrieve information about the mailbox.

To get the information from a mailbox without changing the current mailbox, use [.getBoxInfo\(\)](#).

In the *name* parameter, pass the name of the mailbox to access. The name represents an unambiguous left-to-right hierarchy with levels separated by a specific delimiter character. The delimiter can be found with the [.getDelimiter\(\)](#) function.

The optional *state* parameter defines the type of access to the mailbox. The possible values are:

| Constant                       | Value | Comment   |
|--------------------------------|-------|---|
| IMAP<br>read only              | 1     | The selected mailbox is accessed with read only privileges. Messages with a "recent" flag (indicating new messages) remain unchanged.                                 |
| IMAP<br>read<br>write<br>state | 0     | The selected mailbox is accessed with read and write privileges. Messages are considered "seen" and lose the "recent" flag (indicating new messages). (Default value) |

- The function generates an error and returns **Null** if *name* designates a non-existing mailbox.
- If there is no open connection, `.selectBox()` will open a connection.
- If the connection has not been used since the designated connection delay (see `IMAP New transporter`), the `.checkConnection()` function is automatically called.

## Returned object

The `boxInfo` object returned contains the following properties:

| Property                    | Type   | Description   |
|-----------------------------|--------|---|
| <code>name</code>           | Text   | Name of the mailbox   |
| <code>mailCount</code>      | number | Number of messages in the mailbox   |
| <code>mailRecent</code>     | number | Number of messages with the "recent" flag   |
| <code>id</code>             | text   | Unique id of the mailbox  |
| <code>flags</code>          | text   | List of flags currently used for the mailbox, separated by spaces   |
| <code>permanentFlags</code> | text   | List of flags that the client can change permanently (except for the \$Recent flag, which is managed by the IMAP server), separated by spaces |

### (!) INFO

If `permanentFlags` string includes the special flag \*, it means that the server supports [custom flags](#).

## Example

```
var $server; $boxinfo : Object
$server:=New object
$server.host:="imap.gmail.com" //Mandatory
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXX"

var $transporter : 4D.IMAPTransporter
$transporter:=IMAP New transporter($server)
$boxInfo:=$transporter.selectBox("INBOX")
```

## .subscribe()

- History

**.subscribe( name : Text ) : Object**

| Parameter | Type   | Description                          |
|-----------|--------|--------------------------------------|
| name      | Text   | -> Name of the mailbox               |
| Result    | Object | <- Status of the subscribe operation |

## Description

The `.subscribe()` function allows adding or removing of the specified mailbox to/from the IMAP server's set of "subscribed" mailboxes. As such, you can choose to narrow down a large list of available mailboxes by subscribing to those you usually want to see.

In the `name` parameter, pass the name of the mailbox to add (subscribe) to your "subscribed" mailboxes.

## Returned object

The function returns an object describing the IMAP status:

| Property               | Type       | Description  |
|------------------------|------------|--|
| success                | Boolean    | True if the operation is successful, False otherwise                                     |
| statusText             | Text       | Status message returned by the IMAP server, or last error returned in the 4D error stack |
| errors                 | Collection | 4D error stack (not returned if a IMAP server response is received)                      |
| []].errcode            | Number     | 4D error code  |
| []].message            | Text       | Description of the 4D error  |
| []].componentSignature | Text       | Signature of the internal component which returned the error                             |

## Example

To subscribe to the "Atlas Corp" mailbox in the "Bills" hierarchy:

```

var $pw; $name : text
var $options; $transporter; $status : object

$options:=New object

$pw:=Request("Please enter your password:")

If (OK=1) $options.host:="imap.gmail.com"
$options.user:="test@gmail.com"
$options.password:=$pw

$transporter:=IMAP New transporter($options)

$name:="Bills"+$transporter.getDelimiter()+"Atlas Corp"
$status:=$transporter.subscribe($name)

If ($status.success)
    ALERT("Mailbox subscription successful!")
Else
    ALERT("Error: "+$status.statusText)
End if
End if

```

## .unsubscribe()

- History

**.unsubscribe( name : Text ) : Object**

| Parameter | Type   | Description                            |
|-----------|--------|--|
| name      | Text   | -> Name of the mailbox                 |
| Result    | Object | <- Status of the unsubscribe operation |

### Description

The `.unsubscribe()` function removes a mailbox from a set of subscribed mailboxes. This allows you reduce the number of mailboxes you usually see.

In the `name` parameter, pass the name of the mailbox to remove (unsubscribe) from your active mailboxes.

### Returned object

The function returns an object describing the IMAP status:

| Property               | Type       | Description  |
|------------------------|------------|--|
| success                | Boolean    | True if the operation is successful, False otherwise                                     |
| statusText             | Text       | Status message returned by the IMAP server, or last error returned in the 4D error stack |
| errors                 | Collection | 4D error stack (not returned if a IMAP server response is received)                      |
| [ ].errcode            | Number     | 4D error code  |
| [ ].message            | Text       | Description of the 4D error  |
| [ ].componentSignature | Text       | Signature of the internal component which returned the error                             |

### Example

To unsubscribe from the "Atlas Corp" mailbox in the "Bills" hierarchy:

```

var $pw; $name : text
var $options; $transporter; $status : object

$options:=New object

$pw:=Request("Please enter your password:")

If (OK=1) $options.host:="imap.gmail.com"
$options.user:="test@gmail.com"
$options.password:=$pw

$transporter:=IMAP New transporter($options)

$name:="Bills"+$transporter.getDelimiter()+"Atlas Corp"
$status:=$transporter.unsubscribe($name)

If ($status.success)
  ALERT("Mailbox unsubscription successful!")
Else
  ALERT("Error: "+$status.statusText)
End if
End if

```

## **.user**

- History

**.user** : Text

### **Description**

The **.user** property contains the user name used for authentication on the mail server.

# MailAttachment

Attachment objects allow referencing files within a [Email](#) object. Attachment objects are created using the [MAIL New attachment](#) command.

## Attachment Object

Attachment objects provide the following read-only properties and functions:

- .cid : Text** the ID of the attachment
- .disposition : Text** the value of the `Content-Disposition` header
- .getContent() : 4D.Blob** returns the contents of the attachment object in a `4D.Blob` object
- .name : Text** the name and extension of the attachment
- .path : Text** the POSIX path of the attachment file, if it exists
- .platformPath : Text** the path of the attachment file expressed with the current platform syntax
- .type : Text** the `content-type` of the attachment file

## MAIL New attachment

- History

**MAIL New attachment(** *file* : 4D.File { ; *name* : Text {; *cid* : Text{ ; *type* : Text { ; *disposition* :Text } } } } ) : 4D.MailAttachment

**MAIL New attachment(** *zipFile* : 4D.ZipFile { ; *name* : Text {; *cid* : Text{ ; *type* : Text { ; *disposition* :Text } } } } ) : 4D.MailAttachment

**MAIL New attachment(** *blob* : 4D.Blob { ; *name* : Text {; *cid* : Text{ ; *type* : Text { ; *disposition* :Text } } } } ) : 4D.MailAttachment

**MAIL New attachment(** *path* : Text { ; *name* : Text {; *cid* : Text{ ; *type* : Text { ; *disposition* :Text } } } } ) : 4D.MailAttachment

| Parameter          | Type              | Description   |
|--------------------|-------------------|---|
| <i>file</i>        | 4D.File           | -> Attachment file  |
| <i>zipFile</i>     | 4D.ZipFile        | -> Attachment Zipfile   |
| <i>blob</i>        | 4D.Blob           | -> BLOB containing the attachment                                       |
| <i>path</i>        | Text              | -> Path of the attachment file  |
| <i>name</i>        | Text              | -> Name + extension used by the mail client to designate the attachment |
| <i>cid</i>         | Text              | -> ID of attachment (HTML messages only), or " " if no cid is required  |
| <i>type</i>        | Text              | -> Value of the content-type header                                     |
| <i>disposition</i> | Text              | -> Value of the content-disposition header: "inline" or "attachment".   |
| Result             | 4D.MailAttachment | <- Attachment object  |

## Description

The `MAIL New attachment` command allows you to create an attachment object that you can add to an [Email object](#).

To define the attachment, you can use:

- a *file*, pass a `4D.File` object containing the attachment file.
- a *zipfile*, pass a `4D.ZipFile` object containing the attachment file.
- a *blob*, pass a `4D.Blob` object containing the attachment itself.
- a *path*, pass a **text** value containing the path of the attachment file, expressed with the system syntax. You can pass a full path name or a simple file name (in which case 4D will search for the file in the same directory as the project file).

The optional *name* parameter lets you pass the name and extension to be used by the mail client to designate the attachment. If *name* is omitted and:

- you passed a file path, the name and extension of the file is used,
- you passed a BLOB, a random name without extension is automatically generated.

The optional *cid* parameter lets you pass an internal ID for the attachment. This ID is the value of the `Content-Id` header, it will be used in HTML messages only. The *cid* associates the attachment with a reference defined in the message body using an HTML tag such as `\`. This means that the contents of the attachment (e.g., a picture) should be displayed within the message on the mail client. The final result may vary depending on the mail client. You can pass an empty string in *cid* if you do not want to use this parameter.

You can use the optional *type* parameter to explicitly set the `content-type` of the attachment file. For example, you can pass a string defining a MIME type ("video/mpeg"). This content-type value will be set for the attachment, regardless of its extension. For more information about MIME types, please refer to the [MIME type page on Wikipedia](#).

By default, if the *type* parameter is omitted or contains an empty string, the `content-type` of the attachment file is based on its extension. The following rules are applied for the main MIME types:

| Extension | Content Type                  |
|-----------|-------------------------------|
| jpg, jpeg | image/jpeg                    |
| png       | image/png                     |
| gif       | image/gif                     |
| pdf       | application/pdf               |
| doc       | application/msword            |
| xls       | application/vnd.ms-excel      |
| ppt       | application/vnd.ms-powerpoint |
| zip       | application/zip               |
| gz        | application/gzip              |
| json      | application/json              |
| js        | application/javascript        |
| ps        | application/postscript        |
| xml       | application/xml               |
| htm, html | text/html                     |
| mp3       | audio/mpeg                    |
| other     | application/octet-stream      |

The optional *disposition* parameter lets you pass the `content-disposition` header of the attachment. You can pass one of the following constants from the "Mail" constant theme:

| Constant                      | Value  | Comment  |
|-------------------------------|--------|--|
| mail disposition "attachment" |        | Set the Content-disposition header value to "attachment", which means that the attachment file must be provided as a link in the message.  |
| mail disposition "inline"     | inline | Set the Content-disposition header value to "inline", which means that the attachment must be rendered within the message contents, at the "cid" location. The rendering depends on the mail client. |

By default, if the *disposition* parameter is omitted:

- if the *cid* parameter is used, the `Content-disposition` header is set to "inline",
- if the *cid* parameter is not passed or empty, the `Content-disposition` header is set to "attachment".

## Example 1

You want to send an email with a user-selected file as an attachment and an image embedded in the HTML body:

```
$doc:=Select document(""); "*" ; "Please select a file to attach"; 0
If (OK=1) //If a document was selected

C_OBJECT ($email; $server; $transporter)

$server:=New object
$server.host:="smtp.mail.com"
$server.user:="test_user@mail.com"
$server.password:="p@ssw@rd"
$transporter:=SMTP New transporter ($server)

$email:=New object
$email.from:="test_user@mail.com"
$email.to:="test_user@mail.com"
$email.subject:="This is a test message with attachments"

//add a link to download file
$email.attachments:=New collection(MAIL New attachment (Document))
//insert an inline picture (use a cid)
$email.attachments[1]:=MAIL New
attachment("c:\\Pictures\\4D.jpg"; ""; "4D")

$email.htmlBody:="<html>"+\
"<body>Hello World! "+\
"<img src='cid:4D' >"+\
"</body>"+\
"</head>"+\
"</html>"

$transporter.send ($email) //send mail

End if
```

## Example 2

You want to send an email with a 4D Write Pro area as an attachment:

```

C_BLOB($blob)
WP EXPORT VARIABLE(WPArea;$blob;wk docx)

C_OBJECT($email;$server;$transporter)

$server:=New object
$server.host:="smtp.mail.com"
$server.user:="user@mail.com"
$server.password:="p@ssw@rd"
$transporter:=SMTP New transporter($server)

$email:=New object
$email.from:="user@mail.com"
$email.to:="customer@mail.com"
$email.subject:="New annual report"
$email.textBody:="Please find enclosed our latest annual report."
$email.attachments:=New collection(MAIL New attachment($blob;"Annual
report.docx"))

$transporter.send($email)

```

## 4D.MailAttachment.new()

- History

**4D.MailAttachment.new( file : 4D.File { ; name : Text {; cid : Text{ ; type : Text { ; disposition :Text } } } } ) : 4D.MailAttachment**

**4D.MailAttachment.new( zipFile : 4D.ZipFile { ; name : Text {; cid : Text{ ; type : Text { ; disposition :Text } } } } ) : 4D.MailAttachment**

**4D.MailAttachment.new( blob : 4D.Blob { ; name : Text {; cid : Text{ ; type : Text { ; disposition :Text } } } } ) : 4D.MailAttachment**

**4D.MailAttachment.new( path : Text { ; name : Text {; cid : Text{ ; type : Text { ; disposition :Text } } } } ) : 4D.MailAttachment**

| Parameter   | Type                                 | Description  |
|-------------|--------------------------------------|--|
| file        | 4D.File                              | -> Attachment file   |
| zipFile     | 4D.ZipFile                           | -> Attachment Zipfile  |
| blob        | 4D.Blob                              | -> BLOB containing the attachment  |
| path        | Text                                 | -> Path of the attachment file   |
| name        | Text                                 | -> Name + extension used by the mail client to<br>designate the attachment |
| cid         | Text                                 | -> ID of attachment (HTML messages only), or " " if no<br>cid is required  |
| type        | Text                                 | -> Value of the content-type header  |
| disposition | Text                                 | -> Value of the content-disposition header: "inline" or<br>"attachment".   |
| Result      | 4D.MailAttachment<-Attachment object |  |

### Description

The `4D.MailAttachment.new()` function creates and returns a new object of the `4D.MailAttachment` type. It is identical to the [MAIL New attachment](#) command (shortcut).

### .cid

`.cid : Text`

### Description

The `.cid` property contains the ID of the attachment. This property is used in HTML messages only. If this property is missing, the file is handled as a simple attachment (link).

## .disposition

**.disposition** : Text

### Description

The `.disposition` property contains the value of the `Content-Disposition` header. Two values are available:

- "inline": the attachment is rendered within the message contents, at the "cid" location. The rendering depends on the mail client.
- "attachment": the attachment is provided as a link in the message.

## .getContent()

**.getContent()** : 4D.Blob

### Parameter Type Description

Result      4D.Blob <- Content of the attachment

### Description

The `.getContent()` function returns the contents of the attachment object in a `4D.Blob` object. You can use this method with attachment objects received by the [MAIL Convert from MIME](#) command.

## .name

**.name** : Text

### Description

The `.name` property contains the name and extension of the attachment. By default, it is the name of the file, unless another name was specified in the [MAIL New attachment](#) command.

## .path

**.path** : Text

### Description

The `.path` property contains the POSIX path of the attachment file, if it exists.

## .platformPath

- History

**.platformPath** : Text

### Description

The `.platformPath` property returns the path of the attachment file expressed with the current platform syntax.

## **.type**

**.type** : Text

### **Description**

The `.type` property contains the `content-type` of the attachment file. If this type is not explicitly passed to the [MAIL New attachment](#) command, the `content-type` is based on its file extension.

# POP3 Transporter

The `POP3Transporter` class allows you to retrieve messages from a POP3 email server.

## POP3 Transporter object

POP3 Transporter objects are instantiated with the [POP3 New transporter](#) command. They provide the following properties and functions:

- `.acceptUnsecureConnection : Boolean`**   **True** if 4D is allowed to establish an unencrypted connection
- `.authenticationMode : Text`**   the authentication mode used to open the session on the mail server
- `.checkConnection() : Object`**   checks the connection using information stored in the transporter object
- `.connectionTimeOut : Integer`**   the maximum wait time (in seconds) allowed to establish a connection to the server
- `.delete( msgNumber : Integer )`**   flags the *msgNumber* email for deletion from the POP3 server
- `.getBoxInfo() : Object`**   returns a `boxInfo` object corresponding to the mailbox designated by the [POP3 transporter](#)
- `.getMail( msgNumber : Integer { ; headerOnly : Boolean } ) : Object`**   returns the `Email` object corresponding to the *msgNumber* in the mailbox designated by the [POP3 transporter](#)
- `.getMailInfo( msgNumber : Integer ) : Object`**   returns a `mailInfo` object corresponding to the *msgNumber* in the mailbox designated by the [POP3 transporter](#)
- `.getMailInfoList() : Collection`**   returns a collection of `mailInfo` objects describing all messages in the mailbox designated by the [POP3 transporter](#)
- `.getMIMEAsBlob( msgNumber : Integer ) : Blob`**   returns a BLOB containing the MIME contents for the message corresponding to the *msgNumber* in the mailbox designated by the [POP3 transporter](#)
- `.host : Text`**   the name or the IP address of the host server
- `.logFile : Text`**   the path of the extended log file defined (if any) for the mail connection
- `.port : Integer`**   the port number used for mail transactions
- `.undeleteAll()`**   removes all delete flags set on the emails in the [POP3 transporter](#)
- `.user : Text`**   the user name used for authentication on the mail server

## POP3 New transporter

- History

**POP3 New transporter( server : Object ) : 4D.POP3Transporter**

| Parameter | Type   | Description                      |
|-----------|--|----------------------------------|
| server    | object   | -> Mail server information       |
| Result    | 4D.POP3Transporter<-<br><a href="#">object</a> | <a href="#">POP3 transporter</a> |

## Description

The `POP3 New transporter` command configures a new POP3 connection according to the `server` parameter and returns a new [POP3 transporter](#) object. The returned transporter object will then usually be used to receive emails.

In the `server` parameter, pass an object containing the following properties:

| <i>server</i>  | <b>Default value<br/>(if omitted)</b>                               |
|--|---|
| <b>.acceptUnsecureConnection : Boolean</b> <b>True</b> if 4D is allowed to establish an unencrypted connection   | False   |
| <b>.accessTokenOAuth2</b> : Text   |   |
| <b>.accessTokenOAuth2</b> : Object   |   |
| Text string or token object representing OAuth2 authorization credentials. Used only with OAUTH2 <code>authenticationMode</code> . If <code>accessTokenOAuth2</code> is used but <code>authenticationMode</code> is omitted, the OAuth 2 protocol is used (if allowed by the server). Not returned in <a href="#">SMTP transporter</a> object. | none  |
| <b>.authenticationMode : Text</b> the authentication mode used to open the session on the mail server  | the most secure authentication mode supported by the server is used |
| <b>.connectionTimeOut : Integer</b> the maximum wait time (in seconds) allowed to establish a connection to the server   | 30  |
| <b>.host : Text</b> the name or the IP address of the host server  | <i>mandatory</i>  |
| <b>.logFile : Text</b> the path of the extended log file defined (if any) for the mail connection  | none  |
| <b>.password</b> : Text  |   |
| User password for authentication on the server. Not returned in <a href="#">SMTP transporter</a> object.   | none  |
| <b>.port : Integer</b> the port number used for mail transactions  | 995   |
| <b>.user : Text</b> the user name used for authentication on the mail server   | none  |

## Result

The function returns a [POP3 transporter object](#). All returned properties are **read-only**.

The POP3 connection is automatically closed when the transporter object is destroyed.

## Example

```

var $server : Object
$server:=New object
$server.host:="pop.gmail.com" //Mandatory
$server.port:=995
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXXX"
$server.logFile:="LogTest.txt" //log to save in the Logs folder

var $transporter : 4D.POP3Transporter
$transporter:=POP3 New transporter($server)

$status:=$transporter.checkConnection()
If (Not ($status.success))
    ALERT("An error occurred receiving the mail: "+$status.statusText)
End if

```

## 4D.POP3Transporter.new()

**4D.POP3Transporter.new( server : Object ) : 4D.POP3Transporter**

| Parameter | Type               | Description                                |
|-----------|--------------------|--|
| server    | Object             | -> Mail server information                 |
| Result    | 4D.POP3Transporter | <- <a href="#">POP3 transporter object</a> |

### Description

The `4D.POP3Transporter.new()` function creates and returns a new object of the `4D.POP3Transporter` type. It is identical to the [POP3\\_New\\_transporter](#) command (shortcut).

## .acceptUnsecureConnection

- History

**.acceptUnsecureConnection** : Boolean

### Description

The `.acceptUnsecureConnection` property contains **True** if 4D is allowed to establish an unencrypted connection when encrypted connection is not possible.

It contains **False** if unencrypted connections are unallowed, in which case an error is returned when encrypted connection is not possible.

Available secured ports are:

- SMTP
  - 465: SMTPS
  - 587 or 25: SMTP with STARTTLS upgrade if supported by the server.
- IMAP
  - 143: IMAP non-encrypted port
  - 993: IMAP with STARTTLS upgrade if supported by the server
- POP3
  - 110: POP3 non-encrypted port

- 995: POP3 with STARTTLS upgrade if supported by the server.

## .authenticationMode

- History

**.authenticationMode** : Text

### Description

The `.authenticationMode` property contains the authentication mode used to open the session on the mail server.

By default, the most secured mode supported by the server is used.

Possible values are:

| Value    | Constants                                 | Comment  |
|----------|---|--|
| APOP     | <code>POP3 authentication APOP</code>     | Authentication using APOP protocol (POP3 only) |
| CRAM-MD5 | <code>POP3 authentication CRAM-MD5</code> | Authentication using CRAM-MD5 protocol         |
| LOGIN    | <code>POP3 authentication login</code>    | Authentication using LOGIN protocol            |
| OAuth2   | <code>POP3 authentication OAuth2</code>   | Authentication using OAuth2 protocol           |
| PLAIN    | <code>POP3 authentication plain</code>    | Authentication using PLAIN protocol            |

## .checkConnection()

- History

**.checkConnection()** : Object

| Parameter | Type     | Description                                 |
|-----------|----------|---|
| Result    | Object<- | Status of the transporter object connection |

### Description

The `.checkConnection()` function checks the connection using information stored in the transporter object, recreates the connection if necessary, and returns the status. This function allows you to verify that the values provided by the user are valid and consistent.

### Returned object

The function sends a request to the mail server and returns an object describing the mail status. This object can contain the following properties:

| Property               | Type       | Description  |
|------------------------|------------|--|
| success                | boolean    | True if the check is successful, False otherwise   |
| status                 | number     | (SMTP only) Status code returned by the mail server (0 in case of an issue unrelated to the mail processing) |
| statusText             | text       | Status message returned by the mail server, or last error returned in the 4D error stack                     |
| errors                 | collection | 4D error stack (not returned if a mail server response is received)  |
| [ ].errCode            | number     | 4D error code  |
| [ ].message            | text       | Description of the 4D error  |
| [ ].componentSignature | text       | Signature of the internal component which returned the error   |

## Example

```

var $pw : Text
var $options : Object
$options:=New object

$pw:=Request("Please enter your password:")
if(OK=1)
    $options.host:="pop3.gmail.com"

$options.user:="test@gmail.com"
$options.password:=$pw

$transporter:=POP3 New transporter($options)

$status:=$transporter.checkConnection()
If($status.success)
    ALERT("POP3 connection check successful!")
Else
    ALERT("Error: "+$status.statusText)
End if
End if

```

## .connectionTimeOut

- History

**.connectionTimeOut** : Integer

### Description

The `.connectionTimeOut` property contains the maximum wait time (in seconds) allowed to establish a connection to the server. By default, if the property has not been set in the server object (used to create the transporter object with `SMTP New transporter`, `POP3 New transporter`, or `IMAP New transporter`), the value is 30.

## .delete()

- History

**.delete( msgNumber : Integer )**

| Parameter | Type      | Description                     |
|-----------|-----------|---------------------------------|
| msgNumber | Integer-> | Number of the message to delete |

## Description

The `.delete( )` function flags the *msgNumber* email for deletion from the POP3 server.

In the *msgNumber* parameter, pass the number of the email to delete. This number is returned in the *number* property by the [.getMailInfoList\(\)](#) method.

Executing this method does not actually remove any email. The flagged email will be deleted from the POP3 server only when the `POP3_transporter` object (created with `POP3 New transporter`) is destroyed. The flag could be also be removed using the `.undeleteAll()` method.

If the current session unexpectedly terminates and the connection is closed (e.g., timeout, network failure, etc.), an error message is generated and messages marked for deletion will remain on the POP3 server.

## Example

```
$mailInfoList:=$POP3_transporter.getMailInfoList()
For each($mailInfo;$mailInfoList)
    // Mark your mail as "to be deleted at the end of the session"
    $POP3_transporter.delete($mailInfo.number)
End for each
// Force the session closure to delete the mails marked for deletion
CONFIRM("Selected messages will be deleted.;";"Delete";;"Undo")
If(OK=1) //deletion confirmed
    $POP3_transporter:=Null
Else
    $POP3_transporter.undeleteAll() //remove deletion flags
End if
```

## .getBoxInfo()

- History

**.getBoxInfo()** : Object

| Parameter | Type            | Description |
|-----------|-----------------|-------------|
| Result    | Object<-boxInfo | object      |

## Description

The `.getBoxInfo()` function returns a `boxInfo` object corresponding to the mailbox designated by the [.POP3\\_transporter](#). This function allows you to retrieve information about the mailbox.

The `boxInfo` object returned contains the following properties:

| Property  | Type   | Description                       |
|-----------|--------|-----------------------------------|
| mailCount | Number | Number of messages in the mailbox |
| size      | Number | Message size in bytes             |

## Example

```

var $server; $boxinfo : Object

$server:=New object
$server.host:="pop.gmail.com" //Mandatory
$server.port:=995
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXX"

$transporter:=POP3 New transporter($server)

//mailbox info
$boxInfo:=$transporter.getBoxInfo()
ALERT("The mailbox contains "+String($boxInfo.mailCount)+" messages.")

```

## .getMail()

- History

**.getMail( msgNumber : Integer { ; headerOnly : Boolean } ) : Object**

| Parameter  | Type    | Description   |
|------------|---------|---|
| msgNumber  | Integer | -> Number of the message in the list                          |
| headerOnly | Boolean | -> True to download only the email headers (default is False) |
| Result     | Object  | <- <a href="#">Email object</a>                               |

## Description

The `.getMail()` function returns the `Email` object corresponding to the *msgNumber* in the mailbox designated by the [POP3 transporter](#). This function allows you to locally handle the email contents.

Pass in *msgNumber* the number of the message to retrieve. This number is returned in the `number` property by the [.getMailInfoList\(\)](#) function.

Optionally, you can pass `true` in the *headerOnly* parameter to exclude the body parts from the returned `Email` object. Only headers properties (`headers`, `to`, `from`...) are then returned. This option allows you to optimize the downloading step when a lot of emails are on the server.

### NOTE

The *headerOnly* option may not be supported by the server.

The method returns Null if:

- *msgNumber* designates a non-existing message,
- the message was marked for deletion using [.delete\(\)](#).

## Returned object

`.getMail()` returns an [Email object](#).

## Example

You want to know the sender of the first mail of the mailbox:

```
var $server; $transporter : Object
var $mailInfo : Collection
var $sender : Variant

$server:=New object
$server.host:="pop.gmail.com" //Mandatory
$server.port:=995
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXX"

$transporter:=POP3 New transporter($server)

$mailInfo:=$transporter.getMailInfoList()

$sender:=$transporter.getMail($mailInfo[0].number).from
```

## .getMailInfo()

- History

**.getMailInfo( msgNumber : Integer ) : Object**

| Parameter | Type      | Description                       |
|-----------|-----------|-----------------------------------|
| msgNumber | Integer-> | Number of the message in the list |
| Result    | Object    | <-mailInfo object                 |

## Description

The `.getMailInfo()` function returns a `mailInfo` object corresponding to the *msgNumber* in the mailbox designated by the [POP3 transporter](#). This function allows you to retrieve information about the email.

In *msgNumber*, pass the number of the message to retrieve. This number is returned in the *number* property by the [.getMailInfoList\(\)](#) method.

The `mailInfo` object returned contains the following properties:

| Property | Type   | Description              |
|----------|--------|--------------------------|
| size     | Number | Message size in bytes    |
| id       | Text   | Unique ID of the message |

The method returns **Null** if:

- *msgNumber* designates a non-existing message,
- the message was marked for deletion using `.delete()`.

## Example

```

var $server; $mailInfo : Object
var $mailNumber : Integer

$server.host:="pop.gmail.com" //Mandatory
$server.port:=995
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXX"

var $transporter : 4D.POP3Transporter
$transporter:=POP3 New transporter($server)

//message info
$mailInfo:=$transporter.getMailInfo(1) //get the first mail
If($mailInfo #Null)
    ALERT("First mail size is:"+String($mailInfo.size)+" bytes.")
End if

```

## .getMailInfoList()

- History

### .getMailInfoList() : Collection

| Parameter | Type   | Description |
|-----------|--|-------------|
| Result    | Collection <-Collection of <code>mailInfo</code> objects |             |

### Description

The `.getMailInfoList()` function returns a collection of `mailInfo` objects describing all messages in the mailbox designated by the [POP3 transporter](#). This function allows you to locally manage the list of messages located on the POP3 mail server.

Each `mailInfo` object in the returned collection contains the following properties:

| Property   | Type   | Description  |
|------------|--------|--|
| [ ].size   | Number | Message size in bytes  |
| [ ].number | Number | Message number   |
| [ ].id     | Text   | Unique ID of the message (useful if you store the message locally) |

If the mailbox does not contain a message, an empty collection is returned.

### number and ID properties

*number* is the number of a message in the mailbox at the time the [POP3 transporter](#) was created. The *number* property is not a static value in relation to any specific message and will change from session to session dependent on its relation to other messages in the mailbox at the time the session was opened. The numbers assigned to the messages are only valid during the lifetime of the [POP3 transporter](#). At the time the [POP3 transporter](#) is deleted any message marked for deletion will be removed. When the user logs back into the server, the current messages in the mailbox will be renumbered from 1 to x.

The *id* however is a unique number assigned to the message when it was received by the server. This number is calculated using the time and date that the message is received and is a value assigned by your POP3 server. Unfortunately, POP3 servers do not use the *id* as the primary reference to their messages. Throughout the POP3

sessions you will need to specify the *number* as the reference to messages on the server. Developers may need to take some care if developing solutions which bring references to messages into a database but leave the body of the message on the server.

## Example

You want to know the total number and size of emails in the mailbox:

```
var $server : Object
$server:=New object
$server.host:="pop.gmail.com" //Mandatory
$server.port:=995
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXX"

var $transporter : 4D.POP3Transporter
$transporter:=POP3 New transporter($server)

C_COLLECTION($mailInfo)
C_LONGINT($vNum;$vSize)

$mailInfo:=$transporter.getMailInfoList()
$vNum:=$mailInfo.length
$vSize:=$mailInfo.sum("size")

ALERT("The mailbox contains "+String($vNum)+" message(s) for
"+String($vSize)+" bytes.")
```

## .getMIMEAsBlob()

- History

**.getMIMEAsBlob( msgNumber : Integer ) : Blob**

| Parameter | Type      | Description  |
|-----------|-----------|--|
| msgNumber | Integer-> | Number of the message in the list                        |
| Result    | Blob      | <- Blob of the MIME string returned from the mail server |

## Description

The `.getMIMEAsBlob()` function returns a BLOB containing the MIME contents for the message corresponding to the *msgNumber* in the mailbox designated by the [POP3 transporter](#).

In *msgNumber*, pass the number of the message to retrieve. This number is returned in the *number* property by the [.getMailInfoList\(\)](#) method.

The method returns an empty BLOB if:

- *msgNumber* designates a non-existing message,
- the message was marked for deletion using `.delete()`.

## Returned BLOB

`.getMIMEAsBlob()` returns a BLOB which can be archived in a database or converted to an [Email object](#) with the `MAIL Convert from MIME` command.

## Example

You want to know the total number and size of emails in the mailbox:

```
var $server : Object
var $mailInfo : Collection
var $blob : Blob
var $transporter : 4D.POP3Transporter

$server:=New object
$server.host:="pop.gmail.com"
$server.port:=995
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXX"

$transporter:=POP3 New transporter($server)

$mailInfo:=$transporter.getMailInfoList()
$blob:=$transporter.getMIMEAsBlob($mailInfo[0].number)
```

## .host

- History

**.host** : Text

### Description

The `.host` property contains the name or the IP address of the host server. Used for mail transactions (SMTP, POP3, IMAP).

## .logFile

- History

**.logFile** : Text

### Description

The `.logFile` property contains the path of the extended log file defined (if any) for the mail connection. It can be relative (to the current Logs folder) or absolute.

Unlike regular log files (enabled via the `SET DATABASE PARAMETER` command), extended log files store MIME contents of all sent mails and do not have any size limit. For more information about extended log files, refer to:

- **SMTP connections** - [4DSMTPLog.txt](#)
- **POP3 connections** - [4DPOP3Log.txt](#)
- **IMAP connections** - [4DIMAPLog.txt](#)

## .port

- History

**.port** : Integer

### Description

The `.port` property contains the port number used for mail transactions. By default, if

the *port* property has not been set in the *server* object (used to create the transporter object with `SMTP New transporter`, `POP3 New transporter`, `IMAP New transporter`), the port used is:

- **SMTP** - 587
- **POP3** - 995
- **IMAP** - 993

## .**undeleteAll()**

- History

### .**undeleteAll()**

| Parameter | Type | Description                     |
|-----------|------|---------------------------------|
|           |      | Does not require any parameters |

Does not require any parameters

### **Description**

The `.undeleteAll()` function removes all delete flags set on the emails in the [POP3 transporter](#).

## .**user**

- History

### .**user** : Text

### **Description**

The `.user` property contains the user name used for authentication on the mail server.

## Session

Session objects are returned by the `Session` command when [scalable sessions are enabled in your project](#). The Session object is automatically created and maintained by the 4D web server to control the session of a web client (e.g. a browser). This object provides the web developer with an interface to the user session, allowing to manage privileges, store contextual data, share information between processes, and launch session-related preemptive processes.

For detailed information about the session implementation, please refer to the [web server Sessions](#) section.

### Summary

`.clearPrivileges()` removes all the privileges associated to the session  
`.expirationDate : Text` the expiration date and time of the session cookie  
`.hasPrivilege( privilege : Text ) : Boolean` returns True if the privilege is associated to the session, and False otherwise  
`.idleTimeout : Integer` the inactivity session timeout (in minutes), after which the session is automatically closed by 4D  
`.isGuest() : Boolean` returns True if the session is a Guest session (i.e. it has no privileges)  
`.setPrivileges( privilege : Text )`  
`.setPrivileges( privileges : Collection )`  
`.setPrivileges( settings : Object )` associates the privilege(s) and/or role(s) defined in the parameter to the session  
`.storage : Object` a shared object that can be used to store information available to all requests of the web client  
`.userName : Text` the user name associated to the session

## Session

- History

**Session** : 4D.Session

| Parameter | Type                         | Description |
|-----------|------------------------------|-------------|
| Result    | 4D.Session <- Session object |             |

### Description

The `Session` command returns the `Session` object corresponding to the current scalable user web session.

This command only works when [scalable sessions are enabled](#). It returns `Null` when sessions are disabled or when legacy sessions are used.

When scalable sessions are enabled, the `Session` object is available from any web processes in the following contexts:

- `On Web Authentication`, `On Web Connection`, and `On REST Authentication` database methods,
- [On Mobile App Authentication](#) and [On Mobile App Action](#) database methods for mobile requests,

- ORDA [Data Model Class functions](#) called with REST requests,
- code processed through 4D tags in semi-dynamic pages (4DTEXT, 4DHTML, 4DEVAL, 4DSCRIPT/, 4DCODE)
- project methods with the "Available through 4D tags and URLs (4DACTION...)" attribute and called through 4DACTION/ urls.

## Example

You have defined the `action_Session` method with attribute "Available through 4D tags and URLs". You call the method by entering the following URL in your browser:

```
IP:port/4DACTION/action_Session
//action_Session method
Case of
  :(Session#Null)
    If(Session.hasPrivilege("WebAdmin")) //calling the hasPrivilege
function
  WEB SEND TEXT("4DACTION --> Session is WebAdmin")
Else
  WEB SEND TEXT("4DACTION --> Session is not WebAdmin")
End if
Else
  WEB SEND TEXT("4DACTION --> Session is null")
End case
```

## .clearPrivileges()

- History

### .clearPrivileges()

| Parameter | Type | Description                     |
|-----------|------|---------------------------------|
|           |      | Does not require any parameters |

### Description

The `.clearPrivileges()` function removes all the privileges associated to the session. As a result, the session automatically becomes a Guest session.

## Example

```
//Invalidate a session
var $isGuest : Boolean

Session.clearPrivileges()
$isGuest:=Session.isGuest() // $isGuest is True
```

## .expirationDate

- History

### .expirationDate : Text

### Description

The `.expirationDate` property contains the expiration date and time of the session cookie. The value is expressed as text in the ISO 8601 format: `YYYY-MM-DDTHH:MM:SS.mmmZ`.

This property is **read-only**. It is automatically recomputed if the `.idleTimeout` property value is modified.

## Example

```
var $expiration : Text  
$expiration:=Session.expirationDate //eg "2021-11-05T17:10:42Z"
```

## .hasPrivilege()

- History

**.hasPrivilege( privilege : Text ) : Boolean**

| Parameter | Type    | Description   |
|-----------|---------|---|
| privilege | Text    | <-Name of the privilege to verify                         |
| Result    | Boolean | <- True if session has <i>privilege</i> , False otherwise |

## Description

The `.hasPrivilege()` function returns True if the privilege is associated to the session, and False otherwise.

## Example

You want to check if the "WebAdmin" privilege is associated to the session:

```
If (Session.hasPrivilege("WebAdmin"))  
    //Access is granted, do nothing  
Else  
    //Display an authentication page  
  
End if
```

## .idleTimeout

- History

**.idleTimeout : Integer**

## Description

The `.idleTimeout` property contains the inactivity session timeout (in minutes), after which the session is automatically closed by 4D.

If this property is not set, the default value is 60 (1h).

When this property is set, the `.expirationDate` property is updated accordingly.

The value cannot be less than 60: if a lower value is set, the timeout is raised up to 60.

This property is **read write**.

## Example

```

If (Session.isGuest())
    // A Guest session will close after 60 minutes of inactivity
    Session.idleTimeout:=60
Else
    // Other sessions will close after 120 minutes of inactivity
    Session.idleTimeout:=120
End if

```

## .isGuest()

- History

**.isGuest() : Boolean**

| Parameter | Type       | Description                                     |
|-----------|------------|---|
| Result    | Boolean <- | True if session is a Guest one, False otherwise |

### Description

The `.isGuest()` function returns True if the session is a Guest session (i.e. it has no privileges).

### Example

In the `On Web Connection` database method:

```

If (Session.isGuest())
    //Do something for Guest user
End if

```

## .setPrivileges()

- History

**.setPrivileges( privilege : Text )**  
**.setPrivileges( privileges : Collection )**  
**.setPrivileges( settings : Object )**

| Parameter  | Type       | Description   |
|------------|------------|---|
| privilege  | Text       | -> Privilege name   |
| privileges | Collection | -> Collection of privilege names                              |
| settings   | Object     | -> Object with a "privileges" property (string or collection) |

### Description

The `.setPrivileges()` function associates the privilege(s) and/or role(s) defined in the parameter to the session.

- In the *privilege* parameter, pass a string containing a privilege name (or several comma-separated privilege names).
- In the *privileges* parameter, pass a collection of strings containing privilege names.
- In the *settings* parameter, pass an object containing the following properties:

| Property   | Type               | Description  |
|------------|--------------------|--|
| privileges | Text or Collection | <ul style="list-style-type: none"> <li>• String containing a privilege name, or</li> <li>• Collection of strings containing privilege names</li> </ul> |
| roles      | Text or Collection | <ul style="list-style-type: none"> <li>• String containing a role, or</li> <li>• Collection of strings containing roles</li> </ul>                     |
| userName   | Text               | User name to associate to the session (optional)   |
|            | !                  | INFO   |

Privileges and roles are defined in [roles.json](#) file of the project. For more information, please refer to the [Privileges](#) section.

If the `privileges` or `roles` property contains a name that is not declared in the [roles.json](#) file, it is ignored.

By default when no privilege or role is associated to the session, the session is a [Guest session](#).

The `userName` property is available at session object level (read-only).

## Example

In a custom authentication method, you set the "WebAdmin" privilege to the user:

```
var $userOK : Boolean
...
//Authenticate the user

If ($userOK) //The user has been approved
  var $info : Object
  $info:=New object()
  $info.privileges:=New collection("WebAdmin")
  Session.setPrivileges($info)
End if
```

## .storage

- History

### .storage : Object

#### Description

The `.storage` property contains a shared object that can be used to store information available to all requests of the web client.

When a `Session` object is created, the `.storage` property is empty. Since it is a shared object, this property will be available in the `Storage` object of the server.

Like the `Storage` object of the server, the `.storage` property is always "single": adding a shared object or a shared collection to `.storage` does not create a shared group.

This property is **read only** itself but it returns a read-write object.

## Example

You want to store the client IP in the `.storage` property. You can write in the `On Web Authentication` database method:

```
If (Session.storage.clientIP=NULL) //first access
    Use (Session.storage)
        Session.storage.clientIP:=New shared object("value";
$clientIP)
    End use
End if
```

## .userName

- History

**.userName** : Text

### Description

The `.userName` property contains the user name associated to the session. You can use it to identify the user within your code.

This property is an empty string by default. It can be set using the `privileges` property of the [setPrivileges\(\)](#) function.

This property is **read only**.

# Signal

Signals are tools provided by the 4D language to manage interactions and avoid conflicts between processes in a multiprocess application. Signals allow you to make sure one or more process(es) will wait for a specific task to be completed before continuing execution. Any process can wait and/or release a signal.

Semaphores can also be used to manage interactions. Semaphores allow you to make sure that two or more processes do not modify the same resource (file, record...) at the same time. Only the process that sets the semaphore can remove it.

## Signal Object

A signal is a shared object that must be passed as a parameter to commands that call or create workers or processes.

A `4D.Signal` object contains the following built-in methods and properties:

- `.wait()`
- `.trigger()`
- `.signaled`
- `.description`.

Any worker/process calling the `.wait()` method will suspend its execution until the `.signaled` property is true. While waiting for a signal, the calling process does not use any CPU. This can be very interesting for performance in multiprocess applications. The `.signaled` property becomes true when any worker/process calls the `.trigger()` method.

Note that to avoid blocking situations, the `.wait()` can also return after a defined timeout has been reached.

Signal objects are created with the [New signal](#) command.

## Working with signals

In 4D, you create a new signal object by calling the [New signal](#) command. Once created, this signal must be passed as a parameter to the `New process` or `CALL WORKER` commands so that they can modify it when they have finished the task you want to wait for.

- `signal.wait()` must be called from the worker/process that needs another worker/process to finish a task in order to continue.
- `signal.trigger()` must be called from the worker/process that finished its execution in order to release all others.

Once a signal has been released using a `signal.trigger()` call, it cannot be reused again. If you want to set another signal, you need to call the [New signal](#) command again.

Since a signal object is a [shared object](#), you can use it to return results from called workers/processes, provided that you do not forget to write values within a

Use...End use structure (see example).

## Example

```
var $signal : 4D.Signal

// Creation of a signal
$signal:=New signal

// call main process and execute OpenForm method
CALL WORKER(1;"OpenForm";$signal)
// do another calculation
...
// Waiting for the end of the process
$signaled:=$signal.wait()

// Processing of the results
$calc:=$signal.result+...
```

### **OpenForm** method :

```
#DECLARE ($signal : 4D.Signal)
var $form : Object
$form:=New object("value";0)

// Open the form
$win:=Open form window("Information";Movable form dialog box)
DIALOG("Information";$form)
CLOSE WINDOW($win)

// Add a new attribute to your $signal shared object to pass your
result to the other process:
Use($signal)
$signal.result:=$form.value
End use

// Trigger the signal to the waiting process
$signal.trigger()
```

## Summary

**.description : Text** contains a custom description for the `Signal` object.  
**.signaled : Boolean** contains the current state of the `Signal` object  
**.trigger( )** sets the `signaled` property of the signal object to **true**  
**.wait( { timeout : Real } ) : Boolean** makes the current process wait until the `signaled` property of the signal object to become **true** or the optional `timeout` to expire

## New signal

- History

**New signal { ( description : Text ) } : 4D.Signal**

| Parameter   | Type         | Description                            |
|-------------|--------------|--|
| description | Text         | -> Description for the signal          |
| Result      | 4D.Signal <- | Native object encapsulating the signal |

## Description

The `New signal` command creates a `4D.Signal` object.

A signal is a shared object which can be passed as parameter from a worker or process to another worker or process, so that:

- the called worker/process can update the signal object after specific processing has completed
- the calling worker/process can stop its execution and wait until the signal is updated, without consuming any CPU resources.

Optionally, in the `description` parameter you can pass a custom text describing the signal. This text can also be defined after signal creation.

Since the signal object is a shared object, it can also be used to maintain user properties, including the `.description` property, by calling the `Use...End use` structure.

## Returned value

A new `4D.Signal object`.

## Example

Here is a typical example of a worker that sets a signal:

```
var $signal : 4D.Signal  
$signal:=New signal("This is my first signal")  
  
CALL WORKER("myworker";"doSomething";$signal)  
$signaled:=$signal.wait(1) //wait for 1 second max  
  
If($signaled)  
    ALERT("myworker finished the work. Result: "+$signal.myresult)  
Else  
    ALERT("myworker has not finished in less than 1s")  
End if
```

The `doSomething` method could be like:

```
#DECLARE ($signal : 4D.Signal)  
//any processing  
//...  
Use($signal)  
    $signal.myresult:=$processingResult //return the result  
End use  
$signal.trigger() // The work is finished
```

## .description

- History

**.description** : Text

## Description

The `.description` property contains a custom description for the `Signal` object..

`.description` can be set at the creation of the signal object or at any moment. Note that since the `Signal` object is a shared object, any write-mode access to the `.description` property must be surrounded by a `Use...End use` structure.

This property is **read-write**.

## .signaled

- History

**.signaled** : Boolean

### Description

The `.signaled` property contains the current state of the `Signal` object. When the signal is created, `.signaled` is **False**. It becomes **True** when the `.trigger( )` is called on the object.

This property is **read-only**.

## .trigger()

- History

**.trigger( )**

| Parameter | Type | Description                     |
|-----------|------|---------------------------------|
|           |      | Does not require any parameters |

### Description

The `.trigger( )` function sets the `signaled` property of the signal object to **true** and awakens all workers or processes waiting for this signal.

If the signal is already in the signaled state (i.e., the `signaled` property is already **true**), the function does nothing.

## .wait()

- History

**.wait( { timeout : Real } )** : Boolean

| Parameter | Type    | Description                                       |
|-----------|---------|---|
| timeout   | Real    | -> Maximum waiting time for the signal in seconds |
| Result    | Boolean | <- State of the <code>.signaled</code> property   |

### Description

The `.wait( )` function makes the current process wait until the `.signaled` property of the signal object to become **true** or the optional `timeout` to expire.

To prevent blocking code, you can pass a maximum waiting time in seconds in the `timeout` parameter (decimals are accepted).

**Warning:** Calling `.wait( )` without a `timeout` in the 4D main process is not recommended because it could freeze the whole 4D application.

If the signal is already in the signaled state (i.e. the `signaled` property is already **true**), the function returns immediately, without waiting.

The function returns the value of the `.signaled` property. Evaluating this value allows knowing if the function returned because the `.trigger( )` has been called (`.signaled` is **true**) or if the *timeout* expired (`.signaled` is **false**).

The state of a process that waits for a signal is `Waiting for internal flag`.

## SMTPTransporter

The `SMTPTransporter` class allows you to configure SMTP connections and send emails through *SMTP transporter* objects.

### SMTP Transporter object

SMTP Transporter objects are instantiated with the [SMTP New transporter](#) command. They provide the following properties and functions:

- `.acceptUnsecureConnection : Boolean`**   **True** if 4D is allowed to establish an unencrypted connection
- `.authenticationMode : Text`**   the authentication mode used to open the session on the mail server
- `.bodyCharset : Text`**   the charset and encoding used for the body part of the email
- `.checkConnection() : Object`**   checks the connection using information stored in the transporter object
- `.connectionTimeOut : Integer`**   the maximum wait time (in seconds) allowed to establish a connection to the server
- `.headerCharset : Text`**   the charset and encoding used for the email header
- `.host : Text`**   the name or the IP address of the host server
- `.keepAlive : Boolean`**   **True** if the SMTP connection must be kept alive until the transporter object is destroyed
- `.logFile : Text`**   the path of the extended log file defined (if any) for the mail connection
- `.port : Integer`**   the port number used for mail transactions
- `.send( mail : Object ) : Object`**   sends the *mail object* to the SMTP server defined in the `transporter` object and returns a status object
- `.sendTimeOut : Integer`**   the maximum wait time (in seconds) of a call to `.send()` before a timeout occurs
- `.user : Text`**   the user name used for authentication on the mail server

## SMTP New transporter

- History

**SMTP New transporter**( *server* : Object ) : 4D.SMTPTransporter

| Parameter           | Type  | Description                |
|---------------------|---|----------------------------|
| <code>server</code> | Object  | -> Mail server information |
| Result              | 4D.SMTPTransporter<-<br><a href="#">SMTP transporter object</a> |                            |

### Description

The `SMTP New transporter` command configures a new SMTP connection according to the *server* parameter and returns a new [SMTP transporter](#) object. The returned transporter object will then usually be used to send emails.

This command does not open any connection to the SMTP server. The SMTP connection is actually opened when the `.send()` function is executed.

The SMTP connection is automatically closed:

- when the transporter object is destroyed if the `keepAlive` property is true (default),
- after each `.send()` function execution if the `keepAlive` property is set to false.

In the `server` parameter, pass an object containing the following properties:

| <code>server</code>   | <b>Default value<br/>(if omitted)</b>                               |
|---|---|
| <code>.acceptUnsecureConnection : Boolean</code> <b>True</b> if 4D is allowed to establish an unencrypted connection  | False   |
| <code>.accessTokenOAuth2 : Text</code>  |   |
| <code>.accessTokenOAuth2 : Object</code><br>Text string or token object representing OAuth2 authorization credentials. Used only with OAUTH2 <code>authenticationMode</code> . If <code>accessTokenOAuth2</code> is used but <code>authenticationMode</code> is omitted, the OAuth 2 protocol is used (if allowed by the server). Not returned in <code>SMTP transporter</code> object. | none  |
| <code>.authenticationMode : Text</code> the authentication mode used to open the session on the mail server   | the most secure authentication mode supported by the server is used |
| <code>.bodyCharset : Text</code> the charset and encoding used for the body part of the email   | <code>mail mode</code><br><code>UTF8 (US-ASCII_UTF8_QP)</code>      |
| <code>.connectionTimeOut : Integer</code> the maximum wait time (in seconds) allowed to establish a connection to the server  | 30  |
| <code>.headerCharset : Text</code> the charset and encoding used for the email header   | <code>mail mode</code><br><code>UTF8 (US-ASCII_UTF8_QP)</code>      |
| <code>.host : Text</code> the name or the IP address of the host server   | <i>mandatory</i>  |
| <code>.keepAlive : Boolean</code> <b>True</b> if the SMTP connection must be kept alive until the <code>transporter</code> object is destroyed  | True  |
| <code>.logFile : Text</code> the path of the extended log file defined (if any) for the mail connection   | none  |
| <b>password</b> : Text<br>User password for authentication on the server. Not returned in <code>SMTP transporter</code> object.   | none  |
| <code>.port : Integer</code> the port number used for mail transactions   | 587   |
| <code>.sendTimeOut : Integer</code> the maximum wait time (in seconds) of a call to <code>.send()</code> before a timeout occurs  | 100   |
| <code>.user : Text</code> the user name used for authentication on the mail server  | none  |

## Result

The function returns a **`SMTP transporter object`**. All returned properties are **read-only**.

## Example

```

$server:=New object
$server.host:="smtp.gmail.com" //Mandatory
$server.port:=465
$server.user:="4D@gmail.com"
$server.password:="XXXX"
$server.logFile:="LogTest.txt" //Extended log to save in the Logs
folder

var $transporter : 4D.SMTPTransporter
$transporter:=SMTP New transporter($server)

$email:=New object
$email.subject:="my first mail "
$email.from:="4d@gmail.com"
$email.to:="4d@4d.com;test@4d.com"
$email.textBody:="Hello World"
$email.htmlBody:="<h1>Hello World</h1><h4>'Neque porro quisquam est
qui dolorem ipsum quia dolor sit amet, consectetur, adipisci
velit...</h4>\n
<p>There are many variations of passages of Lorem Ipsum available."\
+"The generated Lorem Ipsum is therefore always free from repetition,
injected humour, or non-characteristic words etc.</p>""

$status:=$transporter.send($email)
If(Not($status.success))
    ALERT("An error occurred sending the mail: "+$status.message)
End if

```

## 4D.SMTPTransporter.new()

**4D.SMTPTransporter.new( server : Object ) : 4D.SMTPTransporter**

| Parameter | Type   | Description                      |
|-----------|--|----------------------------------|
| server    | Object   | -> Mail server information       |
| Result    | 4D.SMTPTransporter<-<br><a href="#">object</a> | <a href="#">SMTP transporter</a> |

### Description

The `4D.SMTPTransporter.new()` function creates and returns a new object of the `4D.SMTPTransporter` type. It is identical to the [SMTP New transporter](#) command (shortcut).

## .acceptUnsecureConnection

- History

**.acceptUnsecureConnection** : Boolean

### Description

The `.acceptUnsecureConnection` property contains **True** if 4D is allowed to establish an unencrypted connection when encrypted connection is not possible.

It contains **False** if unencrypted connections are unallowed, in which case an error is returned when encrypted connection is not possible.

Available secured ports are:

- SMTP

- 465: SMTPS
- 587 or 25: SMTP with STARTTLS upgrade if supported by the server.
- IMAP
  - 143: IMAP non-encrypted port
  - 993: IMAP with STARTTLS upgrade if supported by the server
- POP3
  - 110: POP3 non-encrypted port
  - 995: POP3 with STARTTLS upgrade if supported by the server.

## .authenticationMode

- History

**.authenticationMode** : Text

### Description

The `.authenticationMode` property contains the authentication mode used to open the session on the mail server.

By default, the most secured mode supported by the server is used.

Possible values are:

| Value    | Constants                    | Comment                                |
|----------|------------------------------|--|
| CRAM-MD5 | SMTP authentication CRAM-MD5 | Authentication using CRAM-MD5 protocol |
| LOGIN    | SMTP authentication login    | Authentication using LOGIN protocol    |
| OAuth2   | SMTP authentication OAuth2   | Authentication using OAuth2 protocol   |
| PLAIN    | SMTP authentication plain    | Authentication using PLAIN protocol    |

## .bodyCharset

- History

**.bodyCharset** : Text

### Description

The `.bodyCharset` property contains the charset and encoding used for the body part of the email.

**Possible values:**

| Constant                 | Value                        | Comment   |
|--------------------------|------------------------------|---|
| mail mode ISO2022JP      | US-ASCII_ISO-2022-JP_UTF8_QP | <ul style="list-style-type: none"> <li><i>headerCharset</i>: US-ASCII if possible, Japanese (ISO-2022-JP) &amp; Quoted-printable if possible, otherwise UTF-8 &amp; Quoted-printable</li> <li><i>bodyCharset</i>: US-ASCII if possible, Japanese (ISO-2022-JP) &amp; 7-bit if possible, otherwise UTF-8 &amp; Quoted-printable</li> </ul> |
| mail mode ISO88591       | ISO-8859-1                   | <ul style="list-style-type: none"> <li><i>headerCharset</i>: ISO-8859-1 &amp; Quoted-printable</li> <li><i>bodyCharset</i>: ISO-8859-1 &amp; 8-bit</li> </ul>   |
| mail mode UTF8           | US-ASCII_UTF8_QP             | <i>headerCharset</i> & <i>bodyCharset</i> : US-ASCII if possible, otherwise UTF-8 & Quoted-printable ( <b>default value</b> )   |
| mail mode UTF8 in base64 | US-ASCII_UTF8_B64            | <i>headerCharset</i> & <i>bodyCharset</i> : US-ASCII if possible, otherwise UTF-8 & base64  |

## .checkConnection()

- History

**.checkConnection()** : Object

| Parameter | Type                 | Description                      |
|-----------|----------------------|----------------------------------|
| Result    | Object <- connection | Status of the transporter object |

### Description

The `.checkConnection()` function checks the connection using information stored in the transporter object, recreates the connection if necessary, and returns the status. This function allows you to verify that the values provided by the user are valid and consistent.

### Returned object

The function sends a request to the mail server and returns an object describing the mail status. This object can contain the following properties:

| Property               | Type       | Description  |
|------------------------|------------|--|
| success                | boolean    | True if the check is successful, False otherwise   |
| status                 | number     | (SMTP only) Status code returned by the mail server (0 in case of an issue unrelated to the mail processing) |
| statusText             | text       | Status message returned by the mail server, or last error returned in the 4D error stack                     |
| errors                 | collection | 4D error stack (not returned if a mail server response is received)  |
| [ ].errCode            | number     | 4D error code  |
| [ ].message            | text       | Description of the 4D error  |
| [ ].componentSignature | text       | Signature of the internal component which returned the error   |

For information about SMTP status codes, please refer to [this page](#).

## Example

```
var $pw : Text
var $options : Object
var $transporter : 4D.SMTPTransporter
$options:=New object

$pw:=Request("Please enter your password:")
$options.host:="smtp.gmail.com"

$options.user:="test@gmail.com"
$options.password:=$pw

$transporter:=SMTP New transporter($options)

$status:=$transporter.checkConnection()
If ($status.success=True)
    ALERT("SMTP connection check successful!")
Else
    ALERT("Error # "+String($status.status)+", "+$status.statusText)
End if
```

## .connectionTimeOut

- History

**.connectionTimeOut** : Integer

### Description

The `.connectionTimeOut` property contains the maximum wait time (in seconds) allowed to establish a connection to the server. By default, if the property has not been set in the server object (used to create the transporter object with `SMTP New transporter`, `POP3 New transporter`, or `IMAP New transporter`), the value is 30.

## .headerCharset

- History

**.headerCharset** : Text

### Description

The `.headerCharset` property contains the charset and encoding used for the email header. The header includes the following parts of the email:

- subject,
- attachment filename(s),
- email name.

### Possible values:

| Constant                 | Value                        | Comment   |
|--------------------------|------------------------------|---|
| mail mode ISO2022JP      | US-ASCII_ISO-2022-JP_UTF8_QP | <ul style="list-style-type: none"> <li>• <i>headerCharset</i>: US-ASCII if possible, Japanese (ISO-2022-JP) &amp; Quoted-printable if possible, otherwise UTF-8 &amp; Quoted-printable</li> <li>• <i>bodyCharset</i>: US-ASCII if possible, Japanese (ISO-2022-JP) &amp; 7-bit if possible, otherwise UTF-8 &amp; Quoted-printable</li> </ul> |
| mail mode ISO88591       | ISO-8859-1                   | <ul style="list-style-type: none"> <li>• <i>headerCharset</i>: ISO-8859-1 &amp; Quoted-printable</li> <li>• <i>bodyCharset</i>: ISO-8859-1 &amp; 8-bit</li> </ul>   |
| mail mode UTF8           | US-ASCII_UTF8_QP             | <i>headerCharset</i> & <i>bodyCharset</i> : US-ASCII if possible, otherwise UTF-8 & Quoted-printable (default value)  |
| mail mode UTF8 in base64 | US-ASCII_UTF8_B64            | <i>headerCharset</i> & <i>bodyCharset</i> : US-ASCII if possible, otherwise UTF-8 & base64  |

## .host

- History

**.host** : Text

### Description

The `.host` property contains the name or the IP address of the host server. Used for mail transactions (SMTP, POP3, IMAP).

## .keepAlive

- History

**.keepAlive** : Boolean

### Description

The `.keepAlive` property contains **True** if the SMTP connection must be kept alive until the `transporter` object is destroyed, and **False** otherwise. By default, if the `keepAlive` property has not been set in the `server` object (used to create the `transporter` object with `SMTP New transporter`), it is **True**.

The SMTP connection is automatically closed:

- when the `transporter` object is destroyed if the `.keepAlive` property is true,
- after each `.send( )` function execution if the `.keepAlive` property is set to false.

## .logFile

- History

**.logFile** : Text

### Description

The `.logFile` property contains the path of the extended log file defined (if any) for

the mail connection. It can be relative (to the current Logs folder) or absolute.

Unlike regular log files (enabled via the `SET DATABASE PARAMETER` command), extended log files store MIME contents of all sent mails and do not have any size limit. For more information about extended log files, refer to:

- **SMTP connections** - [4DSMTPLLog.txt](#)
- **POP3 connections** - [4DPOP3Log.txt](#)
- **IMAP connections** - [4DIMAPLog.txt](#)

## .port

- History

**.port** : Integer

### Description

The `.port` property contains the port number used for mail transactions. By default, if the `port` property has not been set in the `server` object (used to create the transporter object with `SMTP New transporter`, `POP3 New transporter`, `IMAP New transporter`), the port used is:

- **SMTP** - 587
- **POP3** - 995
- **IMAP** - 993

## .send()

- History

**.send( *mail* : Object )** : Object

| Parameter | Type                           | Description |
|-----------|--------------------------------|-------------|
| mail      | Object-> <a href="#">Email</a> | to send     |
| Result    | Object<-                       | SMTP status |

### Description

The `.send()` function sends the [mail object](#) to the SMTP server defined in the `transporter` object and returns a status object.

The `transporter` object must have already been created using the `SMTP New transporter` command.

The method creates the SMTP connection if it is not already alive. If the `.keepAlive` property of the `transporter` object is **false**, the SMTP connection is automatically closed after the execution of `.send()`, otherwise it stays alive until the `transporter` object is destroyed. For more information, please refer to the [SMTP New transporter](#) command description.

In `mail`, pass a valid [Email object](#) to send. The origination (where the email is coming from) and destination (one or more recipients) properties must be included, the remaining properties are optional.

### Returned object

The function returns an object describing the SMTP status of the operation. This object can contain the following properties:

| Property   | Type    | Description  |
|------------|---------|--|
| success    | boolean | True if the send is successful, False otherwise  |
| status     | number  | Status code returned by the SMTP server (0 in case of an issue unrelated to the mail processing) |
| statusText | text    | Status message returned by the SMTP server   |

In case of an issue unrelated to the SMTP processing (e.g. a mandatory property is missing in mail), 4D generates an error that you can intercept using a method installed by the `ON ERR CALL` command. Use the `GET LAST ERROR STACK` command for information about the error.

In this case, the resulting status object contains the following values:

| Property   | Value                  |
|------------|------------------------|
| success    | False                  |
| status     | 0                      |
| statusText | "Failed to send email" |

## .sendTimeOut

- History

**.sendTimeOut** : Integer

### Description

The `.sendTimeOut` property contains the maximum wait time (in seconds) of a call to `.send( )` before a timeout occurs. By default, if the `.sendTimeOut` property has not been set in the `server` object, the value 100 is used.

## .user

- History

**.user** : Text

### Description

The `.user` property contains the user name used for authentication on the mail server.

# SystemWorker

System workers allow the 4D code to call any external process (a shell command, PHP, etc.) on the same machine. System workers are called asynchronously. By using callbacks, 4D makes it possible to communicate both ways.

The `SystemWorker` class is available from the `4D` class store.

## Example

```
// Windows example to get access to the ipconfig information
var $myWinWorker : 4D.SystemWorker
var $ipConfig : Text
$myWinWorker:= 4D.SystemWorker.new("ipconfig")
$ipConfig:=$myWinWorker.wait(1).response //timeout 1 second

// macOS example to change the permissions for a file on macOS
// chmod is the macOS command used to modify file access
var $myMacWorker : 4D.SystemWorker
$myMacWorker:= 4D.SystemWorker.new("chmod +x /folder/myfile.sh")
```

## Summary

## **4D.SystemWorker.new ( *commandLine* : Text { ; *options* : Object } ) :**

**4D.SystemWorker** creates and returns a **4D.SystemWorker** object that will execute the *commandLine* you passed as parameter to launch an external process

**.closeInput()** closes the input stream (*stdin*) of the external process

**.commandLine** : Text contains the command line passed as parameter to the **new()** function

**.currentDirectory** : **4D.Folder** contains the working directory in which the external process is executed

**.dataType** : Text contains the type of the response body content

**.encoding** : Text contains the encoding of the response body content

**.errors** : Collection contains a collection of 4D errors in case of execution error(s) if any

**.exitCode** : Integer contains the exit code returned by the external process

**.hideWindow** : Boolean can be used to hide the window of the DOS console or the window of the launched executable (**Windows only**)

**.pid** : Integer contains the process unique identifier of the external process at the system level

**.postMessage( *message* : Text)**

**.postMessage( *messageBLOB* : Blob)** allows you to write on the input stream (*stdin*) of the external process

**.response** : Text

**.response** : Blob contains the concatenation of all data returned once the request is terminated

**.responseError** : Text contains the concatenation of all the errors returned, once the request is terminated

**.terminate()** forces the **SystemWorker** to terminate its execution

**.terminated** : Boolean contains **true** if the external process is terminated

**.timeout** : Integer contains the duration in seconds before the external process will be killed if it is still alive

**.wait( {*timeout* : Real} )** : **4D.SystemWorker** waits until the end of the **SystemWorker** execution or the specified *timeout*

## **4D.SystemWorker.new()**

- History

### **4D.SystemWorker.new ( *commandLine* : Text { ; *options* : Object } ) :**

**4D.SystemWorker**

| <b>Parameter</b> | <b>Type</b>     | <b>Description</b>   |
|------------------|-----------------|--|
| commandLineText  |                 | -> Command line to execute                                       |
| options          | Object          | -> Worker parameters   |
| result           | 4D.SystemWorker | <- New asynchronous System worker or null if process not started |

### **Description**

The **4D.SystemWorker.new()** function creates and returns a **4D.SystemWorker** object that will execute the *commandLine* you passed as parameter to launch an external process.

The returned system worker object can be used to post messages to the worker and get the worker output.

If an error occurs during the creation of the proxy object, the function returns a **null**

object and an error is thrown.

In the `commandLine` parameter, pass the full path of the application's file to be executed (posix syntax), as well as any required arguments, if necessary. If you pass only the application name, 4D will use the `PATH` environment variable to locate the executable.

**Warning:** This function can only launch executable applications; it cannot execute instructions that are part of the shell (command interpreter). For example, under Windows it is not possible to use this command to execute the `dir` instruction.

### ***options* Object**

In the `options` parameter, pass an object that can contain the following properties:

| <b>Property</b>               | <b>Type</b> | <b>Default</b>         | <b>Description</b>   |
|-------------------------------|-------------|------------------------|--|
| <code>onResponse</code>       | Formula     | <code>undefined</code> | Callback for system worker messages. This callback is called once the complete response is received. It receives two objects as parameters (see below)   |
| <code>onData</code>           | Formula     | <code>undefined</code> | Callback for system worker data. This callback is called each time the system worker receives data. It receives two objects as parameters (see below)  |
| <code>onDataError</code>      | Formula     | <code>undefined</code> | Callback for the external process errors ( <code>stderr</code> of the external process). It receives two objects as parameters (see below)   |
| <code>onError</code>          | Formula     | <code>undefined</code> | Callback for execution errors, returned by the system worker in case of unusual runtime conditions (system errors). It receives two objects as parameters (see below)  |
| <code>onTerminate</code>      | Formula     | <code>undefined</code> | Callback when the external process is terminated. It receives two objects as parameters (see below)  |
| <code>timeout</code>          | Number      | <code>undefined</code> | Time in seconds before the process is killed if it is still alive  |
| <code>dataType</code>         | Text        | <code>"text"</code>    | Type of the response body content. Possible values: <code>"text"</code> (default), <code>"blob"</code> . Only if <code>dataType="text"</code> . Encoding of the response body content. For the list of available values, see the <a href="#">CONVERT FROM TEXT</a> command description   |
| <code>variables</code>        | Object      |                        | Sets custom environment variables for the system worker. Syntax: <code>variables.key=value</code> , where <code>key</code> is the variable name and <code>value</code> its value. Values are converted into strings when possible. The value cannot contain a '='. If not defined, the system worker inherits from the 4D environment. |
| <code>currentDirectory</code> | Folder      |                        | Working directory in which the process is executed   |
| <code>hideWindow</code>       | Boolean     | <code>true</code>      | (Windows) Hide the application window (if possible) or the Windows console   |

All callback functions receive two object parameters. Their contents depend on the callback:

| <b>Parameter</b>         | <b>Type</b> | <b>onResponse</b> | <b>onData</b> | <b>onDataError</b> | <b>onError</b> | <b>onTerminate</b> |
|--------------------------|-------------|-------------------|---------------|--------------------|----------------|--------------------|
| \$param1                 | Object      | SystemWorker      | SystemWorker  | SystemWorker       | SystemWorker   | SystemWorker       |
| \$param2.type            | Text        | "response"        | "data"        | "error"            | "error"        | "terminate"        |
| \$param2.data or<br>Blob |             |                   | received data | error data         |                |                    |

Here is the sequence of callback calls:

1. `onData` and `onDataError` are executed one or several times
2. if called, `onError` is executed once (stops the system worker processing)
3. if no error occurred, `onResponse` is executed once
4. `onTerminate` is always executed

## Returned value

The function returns a system worker object on which you can call functions and properties of the `SystemWorker` class.

## Examples on Windows

1. To open Notepad and open a specific document:

```
var $sw : 4D.SystemWorker
var $options : Object
$options:=New object
$options.hideWindow:= False

$sw:=4D.SystemWorker.new ("C:\\WINDOWS\\notepad.exe C:\\Docs\\new
folder\\res.txt";$options)
```

2. Run npm install in the console:

```
var $folder : 4D.Folder
var $options : Object
var $worker : 4D.SystemWorker

$folder:=Folder(fk database folder)
$options:=New object
$options.currentDirectory:=$folder
$options.hideWindow:=False

$worker:=4D.SystemWorker.new("cmd /c npm install";$options)
```

3. To launch the Microsoft® Word® application and open a specific document:

```
$mydoc:="C:\\Program Files\\Microsoft Office\\Office15\\WINWORD.EXE
C:\\Tempo\\output.txt"
var $sw : 4D.SystemWorker
$sw:=4D.SystemWorker.new($mydoc)
```

4. To launch a command with the current directory and post a message:

```

var $param : Object
var $sys : 4D.SystemWorker

$param:=New object
$param.currentDirectory:=Folder(fk database folder)
$sys:=4D.SystemWorker.new("git commit -F -";$param)
$sys.postMessage("This is a postMessage")
$sys.closeInput()

```

### 5. To allow the user to open an external document on Windows:

```

$docname:=Select document ("","","*.*","Choose the file to open";0)
If (OK=1)
  var $sw : 4D.SystemWorker
  $sw:=4D.SystemWorker.new("cmd.exe /C start \"\" \"\"+$docname+\"\"")
End if

```

## Examples on macOS

1. Edit a text file (`cat` is the macOS command used to edit files). In this example, the full access path of the command is passed:

```

var $sw : 4D.SystemWorker
$sw:=4D.SystemWorker.new("/bin/cat /folder/myfile.txt")
$sw.wait() //synchronous execution

```

2. To launch an independent "graphic" application, it is preferable to use the `open` system command (in this case, the code has the same effect as double-clicking the application):

```

var $sw : 4D.SystemWorker
$sw:=4D.SystemWorker.new ("open /Applications/Calculator.app")

```

3. To get the contents of the "Users" folder (`ls -l` is the macOS equivalent of the `dir` command in DOS).

```

var $systemworker : 4D.SystemWorker
var $output : Text
var $errors : Collection

$systemworker:=4D.SystemWorker.new("/bin/ls -l /Users ")
$systemworker.wait(5)
$output:=$systemworker.response
$error:=$systemworker.errors

```

4. Same command as above, but using a sample "Params" user class to show how to handle callback functions:

```

var $systemworker : 4D.SystemWorker
$systemworker:=4D.SystemWorker.new("/bin/ls -l /Users
";cs.Params.new())

// "Params" class

Class constructor
This.dataType:="text"
This.data:=""
This.dataError=""

Function onResponse($systemWorker : Object)
This._createFile("onResponse"; $systemWorker.response)

Function onData($systemWorker : Object; $info : Object)
This.data+=$info.data
This._createFile("onData";this.data)

Function onDataError($systemWorker : Object; $info : Object)
This.dataError+=$info.data
This._createFile("onDataError";this.dataError)

Function onTerminate($systemWorker : Object)
var $textBody : Text
$textBody:="Response: "+$systemWorker.response
$textBody+="ResponseError: "+$systemWorker.responseError
This._createFile("onTerminate"; $textBody)

Function _createFile($title : Text; $textBody : Text)
TEXT TO DOCUMENT(Get 4D folder(Current resources
folder)+$title+".txt"; $textBody)

```

## .closeInput()

- History

### .closeInput()

| Parameter Type | Description                     |
|----------------|---------------------------------|
|                | Does not require any parameters |

### Description

The `.closeInput()` function closes the input stream (`stdin`) of the external process.

When the executable waits for all data to be received through `postMessage()`, `.closeInput()` is useful to indicate to the executable that data sending is finished and that it can proceed.

### Example

```

// Create some data to gzip
var $input;$output : Blob
var $gzip : Text
TEXT TO BLOB("Hello, World!);$input)
$gzip:="\"C:\\Program Files (x86)\\GnuWin32\\bin\\gzip.exe\""

// Create an asynchronous system worker
var $worker : 4D.SystemWorker
$worker:= 4D.SystemWorker.new($gzip;New object("dataType";"blob"))

// Send the compressed file on stdin.
$worker.postMessage($input)
// Note that we call closeInput() to indicate we're done.
// gzip (and most program waiting data from stdin) will wait for more
data until the input is explicitly closed.
$worker.closeInput()
$worker.wait()

$output:=$worker.response

```

## .commandLine

**.commandLine** : Text

### Description

The `.commandLine` property contains the command line passed as parameter to the [new\(\)](#) function.

This property is **read-only**.

## .currentDirectory

**.currentDirectory** : 4D.Folder

### Description

The `.currentDirectory` property contains the working directory in which the external process is executed.

## .dataType

**.dataType** : Text

### Description

The `.dataType` property contains the type of the response body content. Possible values : "text" or "blob".

This property is **read-only**.

## .encoding

**.encoding** : Text

### Description

The `.encoding` property contains the encoding of the response body content. This

property is only available if the `dataType` is "text".

This property is **read-only**.

## .errors

**.errors** : Collection

### Description

The `.errors` property contains a collection of 4D errors in case of execution error(s) if any.

Each element of the collection is an object with the following properties:

| Property                           | Type   | Description  |
|------------------------------------|--------|--|
| <code>[]errorCode</code>           | number | 4D error code  |
| <code>[]message</code>             | text   | Description of the 4D error                                  |
| <code>[ ]componentSignature</code> | text   | Signature of the internal component which returned the error |

If no error occurred, `.errors` is undefined.

## .exitCode

**.exitCode** : Integer

### Description

The `.exitCode` property contains the exit code returned by the external process. If the process did not terminate normally, `exitCode` is *undefined*.

This property is **read-only**.

## .hideWindow

**.hideWindow** : Boolean

### Description

The `.hideWindow` property can be used to hide the window of the DOS console or the window of the launched executable (**Windows only**).

This property is **read-write**.

## .pid

**.pid** : Integer

### Description

The `.pid` property contains the process unique identifier of the external process at the system level.

This property is **read-only**.

## .postMessage()

**.postMessage( message : Text)**  
**.postMessage( messageBLOB : Blob)**

| Parameter   | Type    | Description   |
|-------------|---------|---|
| message     | Text -> | Text to write on the input stream (stdin) of the external process |
| messageBLOB | Blob -> | Bytes write on the input stream                                   |

### Description

The `.postMessage()` function allows you to write on the input stream (stdin) of the external process. In the *message* parameter, pass the text to write in *stdin*.

The `.postMessage()` function also accepts a Blob type value in *messageBLOB* to pass in *stdin*, so that you can post binary data.

You can use the `.dataType` property of the [options object](#) to make response body return Blob values.

### .response

**.response** : Text  
**.response** : Blob

### Description

The `.response` property contains the concatenation of all data returned once the request is terminated, i.e. the full message received from the process output.

The type of the message is defined according to the [dataType](#) attribute.

This property is **read-only**.

### .responseError

**.responseError** : Text

### Description

The `.responseError` property contains the concatenation of all the errors returned, once the request is terminated.

### .terminate()

**.terminate()**

| Parameter | Type | Description                     |
|-----------|------|---------------------------------|
|           |      | Does not require any parameters |

### Description

The `.terminate()` function forces the [SystemWorker](#) to terminate its execution.

This function sends the instruction to terminate and give control back to the executing script.

## .terminated

.terminated : Boolean

### Description

The .terminated property contains **true** if the external process is terminated.

This property is **read-only**.

## .timeout

.timeout : Integer

### Description

The .timeout property contains the duration in seconds before the external process will be killed if it is still alive.

This property is **read-only**.

## .wait()

- History

.wait( {timeout : Real} ) : 4D.SystemWorker

| Parameter | Type                                 | Description                  |
|-----------|--------------------------------------|------------------------------|
| timeout   | Real                                 | -> Waiting time (in seconds) |
| Result    | 4D.SystemWorker<-SystemWorker object |                              |

### Description

The .wait() function waits until the end of the SystemWorker execution or the specified *timeout*.

In *timeout*, pass a value in seconds. The SystemWorker script will wait for the external process for the amount of time defined in the *timeout* parameter. If you omit the *timeout* parameter, the script execution will wait indefinitely.

Actually, .wait() waits until the end of processing of the onTerminate formula, except if the *timeout* is reached. If *timeout* is reached, the SystemWorker is not killed.

During a .wait() execution, callback functions are executed, especially callbacks from other events or from other SystemWorker instances. You can exit from a .wait() by calling [terminate\(\)](#) from a callback.

This function returns the SystemWorker object.

This function is not necessary if you created the SystemWorker from a 4D worker process.

# WebServer

The `WebServer` class API allows you to start and monitor a web server for the main (host) application as well as each hosted component (see the [Web Server object](#) overview). This class is available from the `4D` class store.

## Web Server object

Web server objects are instantiated with the `WEB_Server` command.

They provide the following properties and functions:

### Summary

**`.accessKeyDefined : Boolean`** true if an access key is defined in the settings of the web server

**`.certificateFolder : Text`** folder where the certificate files are located

**`.characterSet : Number`**

**`.characterSet : Text`** character set that the 4D Web Server should use to communicate with browsers connecting to the application

**`.cipherSuite : Text`** cipher list used for the secure protocol

**`.CORSEnabled : Boolean`** CORS (*Cross-origin resource sharing*) service status for the web server

**`.CORSSettings : Collection`** list of allowed hosts and methods for the CORS service

**`.debugLog : Number`** status of the HTTP request log file

**`.defaultHomepage : Text`** name of the default home page

**`.HSTSEnabled : Boolean`** HTTP Strict Transport Security (HSTS) status

**`.HSTSMaxAge : Number`** maximum length of time (in seconds) that HSTS is active for each new client connection

**`.HTTPCompressionLevel : Number`** compression level for all compressed HTTP exchanges for the 4D HTTP server (client requests or server replies)

**`.HTTPCompressionThreshold : Number`** size threshold (bytes) for requests below which exchanges should not be compressed

**`.HTTPEnabled : Boolean`** HTTP protocol state

**`.HTTPPort : Number`** listening IP port number for HTTP

**`.HTTPTrace : Boolean`** activation of `HTTP TRACE`

**`.HTTPSEnabled : Boolean`** HTTPS protocol state

**`.HTTPSPort : Number`** listening IP port number for HTTPS

**`.inactiveProcessTimeout : Number`** life duration (in minutes) of the inactive legacy session processes

**`.inactiveSessionTimeout : Number`** life duration (in minutes) of inactive legacy sessions (duration set in cookie)

**`.IPAddressToListen : Text`** IP address on which the 4D Web Server will receive HTTP requests

**`.isRunning : Boolean`** web server running state

**`.keepSession : Boolean`** `True` if legacy sessions are enabled in the web server, `False` otherwise

**`.logRecording : Number`** log requests (`logweb.txt`) recording value

**`.maxConcurrentProcesses : Number`** maximum number of concurrent web processes supported by the web server

**`.maxRequestSize : Number`** maximum size (in bytes) of incoming HTTP requests (POST) that the web server is allowed to process

**`.maxSessions : Number`** maximum number of simultaneous legacy sessions

**.maxSessions : Number** maximum number of simultaneous legacy sessions  
**.minTLSVersion : Number** minimum TLS version accepted for connections  
**.name : Text** name of the web server application  
**.openSSLVersion : Text** version of the OpenSSL library used  
**.perfectForwardSecrecy : Boolean** PFS availability on the server  
**.rootFolder : Text** path of web server root folder  
**.scalableSession : Boolean** `True` if scalable sessions are used in the web server, and `False` otherwise  
**.sessionCookieDomain : Text** "domain" field of the session cookie  
**.sessionCookieName : Text** name of the cookie used for storing the session ID  
**.sessionCookiePath : Text** "path" field of the session cookie  
**.sessionCookieSameSite : Text** "SameSite" session cookie value  
**.sessionIPAddressValidation : Boolean** IP address validation for session cookies  
**.start() : Object**  
**.start( settings : Object ) : Object** starts the web server on which it is applied  
**.stop()** stops the web server on which it is applied

## WEB Server

- History

**WEB Server** : 4D.WebServer

**WEB Server( option : Integer )** : 4D.WebServer

| Parameter | Type           | Description   |
|-----------|----------------|---|
| option    | Integer        | -> Web server to get (default if omitted = <code>Web server database</code> ) |
| Result    | 4D.WebServer<- | Web server object   |

The `WEB Server` command returns the default Web server object, or the Web server object defined through the *option* parameter.

By default, if the *option* parameter is omitted, the command returns a reference to the Web server of the database, i.e. the default Web server. To designate the Web server to return, you can pass one of the following constants in the *option* parameter:

| Constant                                  | Value | Comment  |
|---|-------|--|
| <code>Web server database</code>          | 1     | Current database Web server (default if omitted)         |
| <code>Web server host database</code>     | 2     | Web server of the host database of a component           |
| <code>Web server receiving request</code> | 3     | Web server that received the request (target Web server) |

The returned Web server object contains the current values of the Web server properties.

### Example

From your component, you want to know if the Web server of the host database is started:

```

// Method of a component
var $hostWS : 4D.WebServer
$hostWS:=WEB Server(Web server host database)
If ($hostWS.isRunning)
  ...
End if
  
```

## WEB Server list

- History

**WEB Server list** : Collection

| Parameter | Type          | Description                                    |
|-----------|---------------|--|
| Result    | Collection <- | Collection of the available Web server objects |

The `WEB Server list` command returns a collection of all Web server objects available in the 4D application.

A 4D application can contain anywhere from one to several Web servers:

- one Web server for the host database (default Web server)
- one Web server for each component.

All available Web servers are returned by the `WEB Server list` command, whether they are actually running or not.

The default Web server object is automatically loaded by 4D at startup. On the other hand, each component Web server that you want to use must be instantiated using the [WEB Server](#) command.

You can use the [.name](#) property of the Web server object to identify the project or component to which each Web server object in the list is attached.

### Example

We want to know how many running web servers are available:

```
var $wSList : Collection
var $vRun : Integer

$wSList:=WEB Server list
$vRun:=$wSList.countValues(True;"isRunning")
ALERT(String($vRun)+" web server(s) running on
"+String($wSList.length)+" available.")
```

## .accessKeyDefined

**.accessKeyDefined** : Boolean

The **.accessKeyDefined** property contains true if an access key is defined in the settings of the web server. This property is used by the WebAdmin web server to validate the security configuration of the administration interface.

## .certificateFolder

**.certificateFolder** : Text

Path of the folder where the certificate files are located. The path is formatted in POSIX full path using filesystems. When using this property in the `settings` parameter of the [.start\(\)](#) function, it can be a [Folder object](#).

## .characterSet

**.characterSet** : Number

**.characterSet** : Text

The character set that the 4D Web Server should use to communicate with browsers connecting to the application. The default value actually depends on the language of the OS. Can be a MIBEnum integer or a Name string, identifiers [defined by IANA](#). Here is the list of identifiers corresponding to the character sets supported by the 4D Web Server:

- 4 = ISO-8859-1
- 12 = ISO-8859-9
- 13 = ISO-8859-10
- 17 = Shift-JIS
- 2024 = Windows-31J
- 2026 = Big5
- 38 = euc-kr
- 106 = UTF-8
- 2250 = Windows-1250
- 2251 = Windows-1251
- 2253 = Windows-1253
- 2255 = Windows-1255
- 2256 = Windows-1256

## **.cipherSuite**

**.cipherSuite** : Text

The cipher list used for the secure protocol. Sets the priority of ciphering algorithms implemented by the 4D web server. Can be a sequence of strings separated by colons (for example "ECDHE-RSA-AES128-..."). See the [ciphers page](#) on the OpenSSL site.

## **.CORSEnabled**

**.CORSEnabled** : Boolean

The CORS (*Cross-origin resource sharing*) service status for the web server. For security reasons, "cross-domain" requests are forbidden at the browser level by default. When enabled (True), XHR calls (e.g. REST requests) from Web pages outside the domain can be allowed in your application (you need to define the list of allowed addresses in the CORS domain list, see `CORSSettings` below). When disabled (False, default), all cross site requests sent with CORS are ignored. When enabled (True) and a non-allowed domain or method sends a cross site request, it is rejected with a "403 - forbidden" error response.

Default: False (disabled)

For more information about CORS, please refer to the [Cross-origin resource sharing page](#) on Wikipedia.

## **.CORSSettings**

**.CORSSettings** : Collection

Contains the list of allowed hosts and methods for the CORS service (see `CORSEnabled` property). Each object must contain a **host** property and, optionally, a **methods** property:

- **host** (text, mandatory): Domain name or IP address from where external pages

are allowed to send data requests to the Server via CORS. Multiple domain attributes can be added to create a white list. If *host* is not present or empty, the object is ignored. Several syntaxes are supported:

- 192.168.5.17:8081
  - 192.168.5.17
  - 192.168.\*
  - 192.168.\*:8081
  - <http://192.168.5.17:8081>
  - [http://\\*.myDomain.com](http://*.myDomain.com)
  - <http://myProject.myDomain.com>
  - \*.myDomain.com
  - myProject.myDomain.com
  - \*
- **methods** (text, optional): Accepted HTTP method(s) for the corresponding CORS host. Separate each method with a ";" (e,g,: "post;get"). If *methods* is empty, null, or undefined, all methods are enabled.

## .debugLog

**.debugLog** : Number

The status of the HTTP request log file (HTTPDebugLog\_nn.txt, stored in the "Logs" folder of the application -- nn is the file number).

- 0 = disabled
- 1 = enabled without body parts (body size is provided in this case)
- 3 = enabled with body parts in response only
- 5 = enabled with body parts in request only
- 7 = enabled with body parts in response and request

## .defaultHomepage

**.defaultHomepage** : Text

The name of the default home page or "" to not send the custom home page.

## .HSTSEnabled

**.HSTSEnabled** : Boolean

The HTTP Strict Transport Security (HSTS) status. HSTS allows the Web server to declare that browsers should only interact with it via secure HTTPS connections. Browsers will record the HSTS information the first time they receive a response from the web server, then any future HTTP requests will automatically be transformed into HTTPS requests. The length of time this information is stored by the browser is specified with the `HSTSMaxAge` property. HSTS requires that HTTPS is enabled on the server. HTTP must also be enabled to allow initial client connections.

## .HSTSMaxAge

**.HSTSMaxAge** : Number

The maximum length of time (in seconds) that HSTS is active for each new client connection. This information is stored on the client side for the specified duration.

Default value: 63072000 (2 years).

## **.HTTPCompressionLevel**

**.HTTPCompressionLevel** : Number

The compression level for all compressed HTTP exchanges for the 4D HTTP server (client requests or server replies). This selector lets you optimize exchanges by either prioritizing speed of execution (less compression) or the amount of compression (less speed).

Possible values:

- 1 to 9 (where 1 is the fastest compression and 9 the highest).
- -1 = set a compromise between speed and rate of compression.

Default = 1 (faster compression).

## **.HTTPCompressionThreshold**

**.HTTPCompressionThreshold** : Number

The size threshold (bytes) for requests below which exchanges should not be compressed. This setting is useful in order to avoid losing machine time by compressing small exchanges.

Default compression threshold = 1024 bytes

## **.HTTPEnabled**

**.HTTPEnabled** : Boolean

The HTTP protocol state.

## **.HTTPPort**

**.HTTPPort** : Number

The listening IP port number for HTTP.

Default = 80

## **.HTTPTrace**

**.HTTPTrace** : Boolean

The activation of `HTTP TRACE`. For security reasons, by default the Web server rejects `HTTP TRACE` requests with an error 405. When enabled, the web server replies to `HTTP TRACE` requests with the request line, header, and body.

## **.HTTPSEnabled**

**.HTTPSEnabled** : Boolean

The HTTPS protocol state.

## **.HTTPSPort**

**.HTTPSPort** : Number

The listening IP port number for HTTPS.

Default = 443

## .inactiveProcessTimeout

### .inactiveProcessTimeout : Number

This property is not returned in [scalable sessions mode](#).

The life duration (in minutes) of the inactive legacy session processes. At the end of the timeout, the process is killed on the server, the `On Web Legacy Close Session` database method is called, then the legacy session context is destroyed.

Default = 480 minutes

## .inactiveSessionTimeout

### .inactiveSessionTimeout : Number

This property is not returned in [scalable sessions mode](#).

The life duration (in minutes) of inactive legacy sessions (duration set in cookie). At the end of this period, the session cookie expires and is no longer sent by the HTTP client.

Default = 480 minutes

## .IPAddressToListen

### .IPAddressToListen : Text

The IP address on which the 4D Web Server will receive HTTP requests. By default, no specific address is defined. Both IPv6 string formats and IPv4 string formats are supported.

## .isRunning

### .isRunning : Boolean

*Read-only property*

The web server running state.

## .keepSession

### .keepSession : Boolean

Contains `True` if legacy sessions are enabled in the web server, `False` otherwise.

## See also

[.scalableSession](#)

## .logRecording

### .logRecording : Number

The log requests (logweb.txt) recording value.

- 0 = Do not record (default)
- 1 = Record in CLF format
- 2 = Record in DLF format
- 3 = Record in ELF format
- 4 = Record in WLF format

## .maxConcurrentProcesses

### .maxConcurrentProcesses : Number

The maximum number of concurrent web processes supported by the web server. When this number (minus one) is reached, 4D will not create any other processes and returns the HTTP status 503 - Service Unavailable to all new requests.

Possible values: 10 - 32000

Default = 100

## .maxRequestSize

### .maxRequestSize : Number

Contains the maximum size (in bytes) of incoming HTTP requests (POST) that the web server is allowed to process. Passing the maximum value (2147483647) means that, in practice, no limit is set. This limit is used to avoid web server saturation due to incoming requests that are too large. If a request reaches this limit, the web server rejects it.

Possible values: 500000 - 2147483647

## .maxSessions

### .maxSessions : Number

This property is not returned in [scalable sessions mode](#).

Contains the maximum number of simultaneous legacy sessions. When you reach the limit, the oldest legacy session is closed (and `On Web Legacy Close Session` database method is called) if the web server needs to create a new one. The number of simultaneous legacy sessions cannot exceed the total number of web processes (`maxConcurrentProcesses` property, 100 by default)

## .minTLSVersion

### .minTLSVersion : Number

The minimum TLS version accepted for connections. Connection attempts from clients supporting only versions below the minimum will be rejected.

Possible values:

- 1 = TLSv1\_0
- 2 = TLSv1\_1
- 3 = TLSv1\_2 (default)
- 4 = TLSv1\_3

If modified, the server must be restarted to use the new value.

## .name

**.name** : Text

*Read-only property*

The name of the web server application.

## **.openSSLVersion**

**.openSSLVersion** : Text

*Read-only property*

The version of the OpenSSL library used.

## **.perfectForwardSecrecy**

**.perfectForwardSecrecy** : Boolean

*Read-only property*

The PFS availability on the server.

## **.rootFolder**

**.rootFolder** : Text

The path of web server root folder. The path is formatted in POSIX full path using filesystems. When using this property in the `settings` parameter, it can be a `Folder` object.

## **.scalableSession**

**.scalableSession** : Boolean

Contains `True` if scalable sessions are used in the web server, and `False` otherwise.

### See also

[.keepSession](#)

## **.sessionCookieDomain**

**.sessionCookieDomain** : Text

The "domain" field of the session cookie. Used to control the scope of the session cookies. If you set, for example, the value `"/*.4d.fr"` for this selector, the client will only send a cookie when the request is addressed to the domain `".4d.fr"`, which excludes servers hosting external static data.

## **.sessionCookieName**

**.sessionCookieName** : Text

The name of the cookie used for storing the session ID.

*Read-only property*

## **.sessionCookiePath**

## .sessionCookiePath : Text

The "path" field of the session cookie. Used to control the scope of the session cookies. If you set, for example, the value "/4DACTION" for this selector, the client will only send a cookie for dynamic requests beginning with 4DACTION, and not for pictures, static pages, etc.

## .sessionCookieSameSite

- History

## .sessionCookieSameSite : Text

The "SameSite" session cookie value. Possible values (using constants):

| Constant | Value    | Description   |
|----------|----------|---|
| Web      |          |   |
| SameSite | "Strict" | <i>Default value</i> - Cookies are only sent in a first-party context   |
| Strict   |          |   |
| Web      |          |   |
| SameSite | "Lax"    | Cookies are also sent on cross-site subrequests but only when a user is navigating to the origin site (i.e. when following a link). |
| Lax      |          |   |
| Web      |          |   |
| SameSite | "None"   | Cookies are sent in all contexts, i.e in responses to both first-party and cross-origin requests.                                   |
| None     |          |   |

See the [Session Cookie SameSite](#) description for detailed information.

## .sessionIPAddressValidation

### .sessionIPAddressValidation : Boolean

This property is not used in [scalable sessions mode](#) (there is no IP address validation).

The IP address validation for session cookies. For security reasons, by default the web server checks the IP address of each request containing a session cookie and rejects it if this address does not match the IP address used to create the cookie. In some specific applications, you may want to disable this validation and accept session cookies, even when their IP addresses do not match. For example when mobile devices switch between WiFi and 3G/4G networks, their IP address will change. In this case, you can allow clients to be able to continue using their web sessions even when the IP addresses change (this setting lowers the security level of your application).

## .start()

- History

### .start() : Object

### .start( *settings* : Object ) : Object

#### Parameter Type

settings    Object-> Web server settings to set at startup

Result      Object <- Status of the web server startup

The `.start()` function starts the web server on which it is applied, using properties set in the optional *settings* object parameter.

The web server starts with default settings defined in the settings file of the project or (host database only) using the `WEB SET OPTION` command. However, using the *settings* parameter, you can define customized properties for the web server session.

All settings of [Web Server objects](#) can be customized, except read-only properties (`.isRunning`, `.name`, `.openSSLVersion`, `.perfectForwardSecrecy`, and `[.sessionCookieName(#sessioncookiename)]`).

Customized session settings will be reset when the `.stop()` function is called.

## Returned object

The function returns an object describing the Web server launch status. This object can contain the following properties:

| Property               | Type       | Description  |
|------------------------|------------|--|
| success                | Boolean    | True if the web server was correctly started, False otherwise        |
| errors                 | Collection | 4D error stack (not returned if the web server started successfully) |
| [ ].errCode            | Number     | 4D error code  |
| [ ].message            | Text       | Description of the 4D error  |
| [ ].componentSignature | Text       | Signature of the internal component which returned the error         |

If the Web server was already launched, an error is returned.

## Example

```
var $settings;$result : Object
var $webServer : 4D.WebServer

$settings:=New
object("HTTPPort";8080;"defaultHomepage";"myAdminHomepage.html")

$webServer:=WEB Server
$result:=$webServer.start($settings)
If($result.success)
  //...
End if
```

## .stop()

- History

### .stop()

| Parameter | Type | Description                     |
|-----------|------|---------------------------------|
|           |      | Does not require any parameters |

The `.stop()` function stops the web server on which it is applied.

If the web server was started, all web connections and web processes are closed, once the currently handled requests are finished. If the web server was not started, the method does nothing.

This function resets the customized web settings defined for the session using the *settings* parameter of the [.start\(\)](#) function, if any.

## **Example**

To stop the database Web server:

```
var $webServer : 4D.WebServer  
  
$webServer:=WEB Server(Web server database)  
$webServer.stop()
```

# WebSocketConnection

- History

The `WebSocketConnection` class API allows you to handle WebSocket connections, once established using the `WebSocketServer` class.

## (!) INFO

For an overview and some examples of the WebSocket server implementation in 4D, please refer to the `WebSocketServer` class.

## WebSocketConnection object

A `WebSocketConnection` object is automatically created when the `WSHandler.onConnection` callback function of the `WebSocketServer` object returns a `connectionHandler` object.

WebSocketConnection objects provide the following properties and functions:

**.handler : Object** the accessor that gets the `connectionHandler` object used to initiate the connection  
**.id : Integer** the unique identifier of the connection  
**.send( message : Text )**  
**.send( message : Blob )**  
**.send( message : Object )** sends a *message* to the client  
**.status : Text** the connection status (can be "Closing", "Closed" or "Connected")  
**.terminate( { code : Integer ; message : Text } )** forces the connection to close  
**.wss : 4D.WebSocketServer** the `WebSocketServer` parent object of the connection

### .handler

**.handler : Object**

#### Description

The `.handler` property contains the accessor that gets the `connectionHandler` object used to initiate the connection.

### .id

**.id : Integer**

#### Description

The `.id` property contains the unique identifier of the connection.

This property is read-only.

### .send()

**.send( message : Text )**  
**.send( message : Blob )**  
**.send( message : Object )**

| Parameter | Type                 | Description            |
|-----------|----------------------|------------------------|
| message   | Text / Blob / Object | -> The message to send |

## Description

The `.send()` function sends a *message* to the client.

The following contents are sent depending on the *message* type:

| Type   | Content  |
|--------|--|
| Text   | Text in UTF-8  |
| Blob   | Binary data  |
| Object | Text in JSON UTF-8 (same result as with <a href="#">JSON.stringify</a> ) |

## .status

`.status` : Text

## Description

The `.status` property contains the connection status (can be "Closing", "Closed" or "Connected").

This property is read-only.

## .terminate()

`.terminate( { code : Integer ; message : Text } )`

| Parameter | Type    | Description   |
|-----------|---------|---|
| code      | Integer | - Error code sent to the client (must be > 3000, otherwise the message is not sent) |
| message   | Text    | - Error message sent to the client  |

## Description

The `.terminate()` function forces the connection to close.

A *code* and *message* can be sent to the client during the closure to indicate the reason of the termination.

## .WSS

`.wss` : 4D.WebSocketServer

## Description

The `.wss` property contains the [WebSocketServer](#) parent object of the connection.

This property is read-only.

# WebSocketServer

## ■ History

The `WebSocketServer` class allows you to create and configure a WebSocket server in 4D. Once the 4D WebSocket server is active, you can open and use WebSocket connections between 4D and clients using the [WebSocketConnection class](#).

## ⓘ ABOUT WEBSOCKET SERVERS

The WebSocket protocol provides full-duplex communication channel between a WebSocket Server and a client (e.g. a Web browser). For more information on WebSocket servers, read [this page on Wikipedia](#).

## ⓘ SEE ALSO

See also [this blog post](#) about the 4D WebSocket server.

## Requirements

To create and handle your WebSocket Server in 4D, you will have to use two 4D build-in classes:

- this class (`4D.WebSocketServer`) to manage the server itself,
- the [4D.WebSocketConnection](#) class to manage connections and messages.

In addition, you will have to create two user classes that will contain callback functions:

- a user class to handle server connections,
- a user class to handle messages.

You must [create the WebSocket server](#) within a [worker](#) to keep the connection alive.

The [4D Web Server](#) must started.

## Example

In this basic example, our WebSocket server will return messages in uppercase.

1. Create the WebSocket server using a worker (mandatory) and pass your server connection class as parameter:

```
//create an instance of the user class
//that will handle the connections to the server
var $handler:cs.myServerHandler
$handler:=cs.myServerHandler.new()

CALL WORKER("WebSocketServer";
Formula(wss:=4D.WebSocketServer.new($handler)))
//assign a variable (wss) to the WebSocket allows you
//to call wss.terminate() afterwards
```

2. Define the `myServerHandler` user class containing callback function(s) used to handle connections to the server:

```
//myServerHandler class

Function onConnection($wss : Object; $event : Object) : Object
    //returns an instance of the user class
    //that will handle the messages
    return cs.myConnectionHandler.new()
```

3. Define the `myConnectionHandler` user class containing callback function(s) used to handle messages:

```
// myConnectionHandler class

Function onMessage($ws : 4D.WebSocketConnection; $message : Object)
    //resends the message in uppercase
    $ws.send(Uppercase($message.data))
```

### CLIENT-SIDE JS

See [this blog post](#) for an example of client-side Javascript code handling a WebSocket connection.

## WebSocketServer object

WebSocket server objects provide the following properties and functions:

- .connections : Collection** all current connections handled by the WebSocket server
- .dataType : Text** the type of the data received or sent
- .handler : Object** the accessor that gets the `WSSHandler` object used to initiate the WebSocket server
- .path : Text** the pattern of the path to access the WebSocket server
- .terminate()** closes the WebSocket server
- .terminated : Boolean** True if the WebSocket server is closed

## 4D.WebSocketServer.new()

**4D.WebSocketServer.new( WSSHandler : Object { ; options : Object } ) :**  
4D.WebSocketServer

| Parameter               | Type               | Description  |
|-------------------------|--------------------|--|
| <code>WSSHandler</code> | Object             | -> Object of the user class declaring the WebSocket Server callbacks |
| <code>options</code>    | Object             | -> WebSocket configuration parameters                                |
| Result                  | 4D.WebSocketServer | <- New WebSocketServer object  |

The `4D.WebSocketServer.new()` function creates and starts a WebSocket server that will use the specified `WSSHandler` callbacks and (optionally) `options`, and returns a `4D.WebSocketServer` object.

Calling this function requires that the [4D Web Server](#) is started. The **host** and **port** of the WebSocket server are the same as the host and port of the 4D Web Server.

### **WSSHandler parameter**

In the `WSSHandler` parameter, pass an instance of a user class that will be called every time an event occurs on the WebSocket server --essentially, connection events. The class should define the following callback functions (only `onConnection` is mandatory):

| <b>Property</b> | <b>Type</b>              | <b>Description</b>   | <b>Default</b> |
|-----------------|--------------------------|--|----------------|
| onConnection    | <a href="#">Function</a> | (mandatory) Callback when a new client connection is started (see below) | undefined      |
| onOpen          | <a href="#">Function</a> | Callback when the WebSocket server is started (see below)                | undefined      |
| onTerminate     | <a href="#">Function</a> | Callback when the WebSocket server is terminated (see below)             | undefined      |
| onError         | <a href="#">Function</a> | Callback when an error has occurred (see below)                          | undefined      |

### **WSHandler.onConnection(WSServer : Object ; event : Object) : Object | null**

| <b>Parameter</b> | <b>Type</b>   | <b>Description</b>   |
|------------------|---|--|
| WSServer         | 4D.WebSocketServer<-Current WebSocket server object |  |
| event            | Object  | <- Parameters  |
| type             | Text  | "connection"   |
| requestObject    |   | <code>request</code> object. Contains information on the connection request (see below)  |
| Function result  | Object  | <code>connectionHandler object</code> (see below). If this function returns a <code>connectionHandler</code> object, a <code>4D.WebSocketConnection object</code> is automatically created and added to the <code>collection of connections</code> . This object is then received as parameter in each function of the <code>connectionHandler</code> object. If the returned value is null or undefined, the connection is cancelled. |

This callback is called when the handshake is complete. It must be called with a valid `connectionHandler object` to create the WebSocket connection, otherwise the connection is cancelled.

### **WSHandler.onOpen(WSServer : Object ; event : Object)**

| <b>Parameter</b> | <b>Type</b>   | <b>Description</b> |
|------------------|---|--------------------|
| WSServer         | 4D.WebSocketServer<-Current WebSocket server object |                    |
| event            | Object  | <- Parameters      |
| typeText         |   | "open"             |

Event emitted when the websocket server is started.

### **WSHandler.onTerminate(WSServer : Object ; event : Object)**

| <b>Parameter</b> | <b>Type</b>   | <b>Description</b> |
|------------------|---|--------------------|
| WSServer         | 4D.WebSocketServer<-Current WebSocket server object |                    |
| event            | Object  | <- Parameters      |
| typeText         |   | "terminate"        |

Event emitted when the HTTP server or the WebSocket server is closed.

### **WSHandler.onError(WSServer : Object ; event : Object)**

| Parameter        | Type  | Description   |
|------------------|---|---|
| WSServer         | 4D.WebSocketServer <- Current WebSocket server object |   |
| event            | Object  | <- Parameters   |
| type             | Text  | "error"<br>Collection of 4D error stack in case of execution error <ul style="list-style-type: none"> <li>• [ ].errorCode (number) - 4D error code</li> <li>• [ ].message (text) - Description of the 4D error</li> <li>• [ ].componentSignature (text) - Signature of the internal component which returned the error</li> </ul> |
| errorsCollection |   |   |

Event emitted when an error occurs on the WebSocket server.

### Example of `wssHandler` class

This example of a basic chat feature illustrates how to handle WebSocket server connections in a `WSSHandler` class.

```
//myWSServerHandler class

Function onConnection($wss : Object; $event : Object) : Object

If (VerifyAddress($event.request.remoteAddress))
    // The VerifyAddress method validates the client address
    // The returned WSConnectionHandler object will be used
    // by 4D to instantiate the 4D.WebSocketConnection object
    // related to this connection
    return cs.myConnectionHandler.new()
        // See connectionHandler object
Else
    // The connection is cancelled
    return Null
End if

Function onOpen($wss : Object; $event : Object)
LogFile("**** Server started")

Function onTerminate($wss : Object; $event : Object)
LogFile("**** Server closed")

Function onError($wss : Object; $event : Object)
LogFile("!!! Server error: "+$event.errors.first().message)
```

### `request` object

A `request` object contains the following properties:

| Parameter     | Type   | Description  |
|---------------|--------|--|
| headers       | Object | The client HTTP GET request. <code>headers.key=value</code> (value can be a collection if the same key appears multiple times)   |
| query         | Object | Object that contains the URL parameters. For example, if parameters are: <code>?key1=value1&amp;key2=value2</code> -><br><code>query.key1=value1, query.key2=value2</code><br>contains only the URL that is present in the actual HTTP |
| url           | Text   | request. Ex: <code>GET /status?name=ryan HTTP/1.1</code> -><br><code>url="/status?name=ryan"</code>  |
| remoteAddress | Text   | IP Address of the client   |

## **connectionHandler object**

As a result of the `WSHandler.onConnection` callback, pass a `connectionHandler` object, which is an instance of a user class that will be called every time an event occurs in the WebSocket connection --essentially, messages received. The class should define the following callback functions (only `onMessage` is mandatory):

| Parameter   | Type                  | Description   |
|-------------|-----------------------|---|
| onMessage   | <code>Function</code> | (mandatory) Function called when a new message is received from this connection |
| onOpen      | <code>Function</code> | Function called when the <code>4D.WebSocketConnection</code> is created         |
| onTerminate | <code>Function</code> | Function called when this connection is terminated                              |
| onError     | <code>Function</code> | Function called when an error occurred  |

**connectionHandler.onMessage(ws : 4D.WebSocketConnection ; event : Object)**

| Parameter | Type                                | Description                            |
|-----------|-------------------------------------|--|
| ws        | <code>4D.WebSocketConnection</code> | <- Current WebSocket connection object |
| event     | Object                              | <- Parameters                          |
|           | type Text                           | "message"                              |
|           | data Text / Blob / Object           | data sent by the client                |

This Callback for WebSocket data. Called each time the WebSocket receives data.

**connectionHandler.onOpen(ws : 4D.WebSocketConnection ; event : Object)**

| Parameter | Type                                | Description                            |
|-----------|-------------------------------------|--|
| ws        | <code>4D.WebSocketConnection</code> | <- Current WebSocket connection object |
| event     | Object                              | <- Parameters                          |
|           | typeText                            | "open"                                 |

Called when the `connectionHandler` object is created (after `WSS.onConnection` event).

**connectionHandler.onTerminate(ws : 4D.WebSocketConnection ; event : Object)**

| Parameter | Type                                | Description  |
|-----------|-------------------------------------|--|
| ws        | <code>4D.WebSocketConnection</code> | <- Current WebSocket connection object   |
| event     | Object                              | <- Parameters  |
|           | type Text                           | "terminate"  |
|           | code Number                         | Status code indicating why the connection has been closed. If the WebSocket does not return an error code, <code>code</code> is set to 1005 if no error occurred or to 1006 if there was an error. |
|           | reasonText                          | String explaining why the connection has been closed. If the websocket doesn't return a reason, code is undefined  |

Function called when the WebSocket is closed.

**connectionHandler.onError(ws : 4D.WebSocketConnection ; event : Object)**

| Parameter        | Type                                   | Description  |
|------------------|--|--|
| ws               | <a href="#">4D.WebSocketConnection</a> | <- Current WebSoc connecti object  |
| event            | Object                                 | <- Parameter   |
| type             | Text                                   | "error"  |
| errorsCollection |  | <p>Collection of 4D errors stack in case of execution error</p> <ul style="list-style-type: none"> <li>• [ ].errCode (number) - 4D error code</li> <li>• [ ].message (text) - Description of the 4D error</li> <li>• [ ].componentSignature (text) - Signature of the internal component which returned the error</li> </ul> |

Function called when an error has occurred.

### Example of `connectionHandler` class

This example of a basic chat feature illustrates how to handle messages in a `connectionHandler` class.

```
// myConnectionHandler Class

Function onMessage($ws : 4D.WebSocketConnection; $message : Object)
    // Resend the message to all chat clients
    This.broadcast($ws;$message.data)

Function onOpen($ws : 4D.WebSocketConnection; $message : Object)
    // Send a message to new connected users
    $ws.send("Welcome on the chat!")
    // Send "New client connected" message to all other chat clients
    This.broadcast($ws;"New client connected")

Function onTerminate($ws : 4D.WebSocketConnection; $message : Object)
    // Send "Client disconnected" message to all other chat clients
    This.broadcast($ws;"Client disconnected")

Function broadcast($ws : 4D.WebSocketConnection; $message:text)
    var $client:4D.WebSocketConnection
    // Resend the message to all chat clients
    For each ($client; $ws.wss.connections)
        // Check that the id is not the current connection
        If ($client.id#$ws.id)
            $client.send($message)
        End if
    End for each
```

### ***options* parameter**

In the optional *options* parameter, pass an object that contains the following properties:

| <b>Property Type</b> | <b>Description</b>  | <b>Default</b> |
|----------------------|---|----------------|
| path Text            | Represents the path to access the WebSocket server. If no path is defined, the WebSocket server manages all the connections   | undefined      |
| dataType Text        | Type of the data received through the <code>connectionHandler.onMessage</code> and the data send by <code>WebSocketConnection.send()</code> function. Values: "text", "blob", "object"). If "object": (send) transforms object into a json format and sends it; (reception): receives json format and transforms it into object | text           |

## .connections

**.connections** : Collection

### Description

The `.connections` property contains all current connections handled by the WebSocket server. Each element of the collection is a [WebSocketConnection object](#).

When a connection is terminated, its `status` changes to "Closed" and it is removed from this collection.

## .dataType

**.dataType** : Text

### Description

The `.dataType` property contains the type of the data received or sent.

This property is read-only.

## .handler

**.handler** : Object

### Description

The `.handler` property contains the accessor that gets the `WSSHandler` object used to initiate the WebSocket server.

## .path

**.path** : Text

### Description

The `.path` property contains the pattern of the path to access the WebSocket server. If no path was defined, the WebSocket server manages all connections.

This property is read-only.

## .terminate()

**.terminate()**

| Parameter Type | Description                     |
|----------------|---------------------------------|
|                | Does not require any parameters |

## Description

The `.terminate()` function closes the WebSocket server.

## **.terminated**

**.terminated** : Boolean

## Description

The `.terminated` property contains True if the WebSocket server is closed.

This property is read-only.

## ZIPArchive

A 4D ZIP archive is a `File` or `Folder` object containing one or more files or folders, which are compressed to be smaller than their original size. These archives are created with a ".zip" extension and can be used to save disk space or transfer files via mediums which may have size limitations (e.g., email or network).

- You create a 4D ZIP archive with the [ZIP Create archive](#) command.
- 4D `ZIPFile` and `ZIPFolder` instances are available through the `root` property (`ZIPFolder`) of the object returned by [ZIP Read archive](#) command.

### Example

To retrieve and view the contents of a ZIP file object:

```
var $path; $archive : 4D.File
var $zipFile : 4D.ZipFile
var $zipFolder : 4D.ZipFolder
var $txt : Text

$path:=Folder(fk desktop folder).file("MyDocs/Archive.zip")
$archive:=ZIP Read archive($path)
$zipFolder:=$archive.root // store the zip main folder
$zipFile:=$zipFolder.files()[0] //read the first zipped file

If($zipFile.extension=".txt")
  $txt:=$zipFile.getText()
End if
```

### Summary

[\*\*.root : 4D.ZipFolder\*\*](#) a virtual folder providing access to the contents of the ZIP archive

## ZIP Create archive

- History

**ZIP Create archive** ( *fileToZip* : 4D.File ; *destinationFile* : 4D.File ) : Object

**ZIP Create archive** ( *folderToZip* : 4D.Folder ; *destinationFile* : 4D.File { ; *options* : Integer } ) : Object

**ZIP Create archive** ( *zipStructure* : Object ; *destinationFile* : 4D.File ) : Object

| Parameter              | Type      | Description   |
|------------------------|-----------|---|
| <i>fileToZip</i>       | 4D.File   | -> File or Folder object to compress                                    |
| <i>folderToZip</i>     | 4D.Folder | -> File or Folder object to compress                                    |
| <i>zipStructure</i>    | Object    | -> File or Folder object to compress                                    |
| <i>destinationFile</i> | 4D.File   | -> Destination file for the archive                                     |
| <i>options</i>         | Integer   | -> <i>folderToZip</i> option: <code>ZIP Without enclosing folder</code> |
| <i>Result</i>          | Object    | <- Status object  |

### Description

The `ZIP Create archive` command creates a compressed ZIP archive object and returns the status of the operation.

You can pass a 4D.File, a 4D.Folder, or a zip structure object as first parameter:

- *fileToZip*: You simply pass a 4D.File to compress.
- *folderToZip*: You pass a 4D.Folder to compress. In this case, the *options* parameter allows you to compress only the contents of the folder (i.e., exclude the enclosing folder). By default, ZIP Create archive will compress the folder and its contents, so that the decompressing operation will recreate a folder. If you want the decompressing operation to restore only the contents of the folder, pass the ZIP Without enclosing folder constant in the *options* parameter.
- *zipStructure*: You pass an object describing the ZIP archive object. The following properties are available to define the structure:

| Property           | Type                 | Description  |          |      |             |        |                      |                |                 |  |  |        |        |  |
|--------------------|----------------------|--|----------|------|-------------|--------|----------------------|----------------|-----------------|--|--|--------|--------|--|
| compressionInteger |                      | <ul style="list-style-type: none"><li>• ZIP Compression standard: Deflate compression (default)</li><li>• ZIP Compression LZMA: LZMA compression</li><li>• ZIP Compression XZ: XZ compression</li><li>• ZIP Compression none: No compression</li></ul>   |          |      |             |        |                      |                |                 |  |  |        |        |  |
| level              | Integer              | Compression level. Possible values: 1 to 10. A lower value will produce a larger file, while a higher value will produce a smaller file. Compression level has however an impact on performance. Default values if omitted: <ul style="list-style-type: none"><li>• ZIP Compression standard: 6</li><li>• ZIP Compression LZMA: 4</li><li>• ZIP Compression XZ: 4</li></ul>  |          |      |             |        |                      |                |                 |  |  |        |        |  |
| encryption         | Integer              | The encryption to use if a password is set: <ul style="list-style-type: none"><li>• ZIP Encryption AES128: AES encryption using 128-bit key.</li><li>• ZIP Encryption AES192: AES encryption using 192-bit key.</li><li>• ZIP Encryption AES256: AES encryption using 256-bit key (default if password is set).</li><li>• ZIP Encryption none: Data is not encrypted (default if no password is set)</li></ul>   |          |      |             |        |                      |                |                 |  |  |        |        |  |
| password           | Text                 | A password to use if encryption is required. <ul style="list-style-type: none"><li>• a collection of 4D.File or 4D.Folder objects or</li><li>• a collection of objects with the following properties:<table><thead><tr><th>Property</th><th>Type</th><th>Description</th></tr></thead><tbody><tr><td>source</td><td>4D.File or 4D.Folder</td><td>File or Folder</td></tr><tr><td>destinationText</td><td></td><td>(optional) - Specify a relative filepath to change the organization of the contents of the archive</td></tr><tr><td>option</td><td>number</td><td>(optional) - ZIP Ignore invisible files or 0 to compress all of the file</td></tr></tbody></table></li></ul> | Property | Type | Description | source | 4D.File or 4D.Folder | File or Folder | destinationText |  | (optional) - Specify a relative filepath to change the organization of the contents of the archive | option | number | (optional) - ZIP Ignore invisible files or 0 to compress all of the file |
| Property           | Type                 | Description  |          |      |             |        |                      |                |                 |  |  |        |        |  |
| source             | 4D.File or 4D.Folder | File or Folder   |          |      |             |        |                      |                |                 |  |  |        |        |  |
| destinationText    |                      | (optional) - Specify a relative filepath to change the organization of the contents of the archive   |          |      |             |        |                      |                |                 |  |  |        |        |  |
| option             | number               | (optional) - ZIP Ignore invisible files or 0 to compress all of the file   |          |      |             |        |                      |                |                 |  |  |        |        |  |
| callback           | 4D.Function          | A callback formula that will receive the compression progress (0 - 100) in \$1.  |          |      |             |        |                      |                |                 |  |  |        |        |  |

In the *destinationFile* parameter, pass a 4D.File object describing the ZIP archive to create (name, location, etc.). It is advised to use the ".zip" extension if you want the ZIP archive to be processed automatically by any software.

Once an archive is created, you can use the [ZIP Read archive](#) command to access it.

## Status object

The returned status object contains the following properties:

| Property   | Type    | Description   |
|------------|---------|---|
| statusText | Text    | Error message (if any): <ul style="list-style-type: none"><li>• Cannot open ZIP archive</li><li>• Cannot create ZIP archive</li><li>• Password is required for encryption</li></ul> |
| status     | Integer | Status code   |
| success    | Boolean | True if archive created successfully, else false  |

### Example 1

To compress a `4D.File`:

```
var $file; $destination : 4D.File
var $status : Object

$destination:=Folder(fk desktop folder).file("MyDocs/file.zip")
$file:=Folder(fk desktop folder).file("MyDocs/text.txt")

$status:=ZIP Create archive($file;$destination)
```

### Example 2

To compress a `4D.Folder` without the folder itself:

```
var $folder : 4D.Folder
var $destination : 4D.File
var $status : Object

$destination:=Folder(fk desktop folder).file("MyDocs/Images.zip")
$folder:=Folder(fk desktop folder).folder("MyDocs/Images")

$status:=ZIP Create archive($folder;$destination;ZIP Without enclosing
folder)
```

### Example 3

To compress a ZIP archive structure with a password and progress bar:

```
var $destination : 4D.File
var $zip;$status : Object
var progID : Integer

$destination:=Folder(fk desktop folder).file("MyDocs/Archive.zip")

$zip:=New object
$zip.files:=Folder(fk desktop
folder).folder("MyDocs/Resources").folders()
$zip.password:="password"
$zip.callback:=Formula(myFormulaCompressingMethod($1))

progID:=Progress New //we use the 4D Progress component

$status:=ZIP Create archive($zip;$destination)

Progress QUIT(progID)
```

```
myFormulaCompressingMethod:
```

```
var $1 : Integer  
Progress SET PROGRESS(progID;Num($1/100))
```

## Example 4

You want to pass a collection of folders and files to compress to the *zipStructure* object:

```
var $destination : 4D.File  
var $zip;$err : Object  
$zip:=New object  
$zip.files:=New collection  
$zip.files.push(New object("source";Folder(fk desktop  
folder).file("Tests/text.txt")))  
$zip.files.push(New object("source";Folder(fk desktop  
folder).file("Tests/text2.txt")))  
$zip.files.push(New object("source";Folder(fk desktop  
folder).file("Images/image.png")))  
  
$destination:=Folder(fk desktop folder).file("file.zip")  
$err:=ZIP Create archive($zip;$destination)
```

## Example 5

You want to use an alternative compression algorithm with a high compression level:

```
var $destination : 4D.File  
var $zip; $err : Object  
  
$zip:=New object  
$zip.files:=New collection  
$zip.files.push(Folder(fk desktop folder).folder("images"))  
$zip.compression:=ZIP Compression LZMA  
$zip.level:=7 //default is 4  
  
$destination:=Folder(fk desktop folder).file("images.zip")  
$err:=ZIP Create archive($zip; $destination)
```

## ZIP Read archive

- History

**ZIP Read archive** (*zipFile* : 4D.File { ; *password* : Text }) : 4D.ZipArchive

| Parameter | Type          | Description                    |
|-----------|---------------|--------------------------------|
| zipFile   | 4D.File       | -> Zip archive file            |
| password  | Text          | -> ZIP archive password if any |
| Result    | 4D.ZipArchive | <- Archive object              |

### Description

The `ZIP Read archive` command retrieves the contents of *zipFile* and returns it as a `4D.ZipArchive` object.

This command does not uncompress the ZIP archive, it only provides a view of its contents. To extract the contents of an archive, you need to use methods such as [file.copyTo\(\)](#) or [folder.copyTo\(\)](#).

Pass a `4D.File` object referencing the compressed ZIP archive in the `zipFile` parameter. The target archive file will be opened until the `ZIP Read archive` has finished executing and all contents/references have been extracted/released, then it will be closed automatically.

If the `zipFile` is password protected, you need to use the optional `password` parameter to provide a password. If a password is required but not passed when trying to read the contents of the archive, an error is generated.

## Archive object

The returned `4D.ZipArchive` object contains a single `root` property whose value is a `4D.ZipFolder` object. This folder describes the whole contents of the ZIP archive.

### Example

To retrieve and view the contents of a ZIP file object:

```
var $archive : 4D.ZipArchive
var $path : 4D.File

$path:=Folder(fk desktop folder).file("MyDocs/Archive.zip")
$archive:=ZIP Read archive($path)
```

To retrieve the list of the files and folders in the archive:

```
$folders:=$archive.root.folders()
$files:=$archive.root.files()
```

To read the contents of a file without extracting it from the root folder:

```
If($files[$i].extension=".txt")
    $txt:=$files[$i].getText()
Else
    $blob:=$files[$i].getContent()
End if
```

To extract from the root folder:

```
//extract a file
$folderResult:=$files[$i].copyTo(Folder(fk desktop
folder).folder("MyDocs"))

//extract all files
$folderResult:=$archive.root.copyTo(Folder(fk desktop
folder).folder("MyDocs"))
```

## .root

**.root** : `4D.ZipFolder`

### Description

The `.root` property contains a virtual folder providing access to the contents of the ZIP archive.

The `root` folder and its contents can be manipulated with the [ZipFile](#) and [ZipFolder](#) functions and properties.

This property is **read-only**.



## ZIPFile

The following properties and functions from the [File](#) class are available to `ZIPFile` objects:

| Available <a href="#">File APIs</a> for ZIPFile   | Comment                                |
|---|--|
| <code>.copyTo( destinationFolder : 4D.Folder { ; newName : Text }<br/>{} ; overwrite : Integer ) : 4D.File</code> |  |
| <code>.creationDate : Date</code>   |  |
| <code>.creationTime : Time</code>   |  |
| <code>.exists : Boolean</code>  |  |
| <code>.extension : Text</code>  |  |
| <code>.fullName : Text</code>   |  |
| <code>.getContent( ) : 4D.Blob</code>   |  |
| <code>.getIcon( { size : Integer } ) : Picture</code>   |  |
| <code>.getText( { charSetName : Text { ; breakMode : Integer } } )<br/>: Text</code>                              |  |
| <code>.getText( { charSetNum : Integer { ; breakMode : Integer } }<br/>) : Text</code>                            |  |
| <code>.hidden : Boolean</code>  |  |
| <code>.isAlias : Boolean</code>   |  |
| <code>.isFile : Boolean</code>  |  |
| <code>.isFolder : Boolean</code>  |  |
| <code>.isWritable : Boolean</code>  | Always false with ZIP archive          |
| <code>.modificationDate : Date</code>   |  |
| <code>.modificationTime : Time</code>   |  |
| <code>.name : Text</code>   |  |
| <code>.original : 4D.File</code>  |  |
| <code>.original : 4D.Folder</code>  |  |
| <code>.parent : 4D.Folder</code>  |  |
| <code>.path : Text</code>   | Returns a path relative to the archive |
| <code>.platformPath : Text</code>   |  |

## ZIPFolder

The following properties and functions from the [Folder](#) class are available to `ZIPFolder` objects:

| Available <a href="#">Folder APIs</a> for ZIPFolder  | Comment  |
|--|--|
| <a href="#"><code>.copyTo( destinationFolder : 4D.Folder { ; newName : Text } { ; overwrite : Integer } ) : 4D.Folder</code></a> |  |
| <a href="#"><code>.creationDate : Date</code></a>  | Date may be different for the <code>root</code> folder from a folder within the archive  |
| <a href="#"><code>.creationTime : Time</code></a>  | Time may be different for the <code>root</code> folder from a folder within the archive  |
| <a href="#"><code>.exists : Boolean</code></a>   |  |
| <a href="#"><code>.extension : Text</code></a>   |  |
| <a href="#"><code>.file( path : Text ) : 4D.File</code></a>  |  |
| <a href="#"><code>.files( { options : Integer } ) : Collection</code></a>  |  |
| <a href="#"><code>.folder( path : Text ) : 4D.Folder</code></a>  |  |
| <a href="#"><code>.folders( { options : Integer } ) : Collection</code></a>  |  |
| <a href="#"><code>.fullName : Text</code></a>  |  |
| <a href="#"><code>.getIcon( { size : Integer } ) : Picture</code></a>  |  |
| <a href="#"><code>.hidden : Boolean</code></a>   |  |
| <a href="#"><code>.isAlias : Boolean</code></a>  |  |
| <a href="#"><code>.isFile : Boolean</code></a>   |  |
| <a href="#"><code>.isFolder : Boolean</code></a>   |  |
| <a href="#"><code>.isPackage : Boolean</code></a>  |  |
| <a href="#"><code>.modificationDate : Date</code></a>  | Date may be different for the <code>root</code> folder from a folder within the archive  |
| <a href="#"><code>.modificationTime : Time</code></a>  | Time may be different for the <code>root</code> folder from a folder within the archive  |
| <a href="#"><code>.name : Text</code></a>  |  |
| <a href="#"><code>.original : 4D.Folder</code></a>   |  |
| <a href="#"><code>.parent : 4D.Folder</code></a>   | The archive's virtual <code>root</code> folder has no parent. However, the folders within the archive may have a parent other than the root. |
| <a href="#"><code>.path : Text</code></a>  | Returns a path relative to the archive   |
| <a href="#"><code>.platformPath : Text</code></a>  |  |

## Handling Code

The 4D code used across your application is written in [methods](#) and [classes](#).

The 4D IDE provides you with various features to create, edit, export, or delete your code. You will usually use the 4D [code editor](#) to work with your code.

### Creating methods

A method in 4D is stored in a **.4dm** file located in the appropriate folder of the [/Project/Sources/](#) folder.

You can create [several types of methods](#):

- All types of methods can be created or opened from the **Explorer** window (except Object methods which are managed from the [Form editor](#)).
- Project methods can also be created or opened from the **File** menu or toolbar (**New/Method...** or **Open/Method...**) or using shortcuts in the [Code editor window](#).
- Triggers can also be created or opened from the Structure editor.
- Form methods can also be created or opened from the [Form editor](#).

### Creating classes

A user class in 4D is defined by a specific method file (**.4dm**), stored in the [/Project/Sources/Classes/](#) folder. The name of the file is the class name.

You can create a class file from the **File** menu or toolbar (**New/Class...**) or in the **Methods** page of the **Explorer** window.

For more information, please refer to the [Classes](#) section.

### Deleting methods or classes

To delete an existing method or class, you can:

- on your disk, remove the **.4dm** file from the "Sources" folder,
- in the 4D Explorer, select the method or class and click or choose **Move to Trash** from the contextual menu.

To delete an object method, choose **Clear Object Method** from the [Form editor](#) (**Object** menu or context menu).

### Importing and exporting code

You can import and export a method or a class code in the form of a file. These commands are found in the **Method** menu of the Code editor.

- When you select the **Export Method...** command, a standard file saving dialog box appears, allowing you to choose the name, location and format of the export file (see below). As with printing, exporting does not take the collapsed state of code structures into account and the entire code is exported.
- When you select the **Import Method...** command, a standard file opening dialog box appears, allowing you to designate the file to be imported. Importing replaces the selected text in the method. To replace an existing method by an imported method, select the entire contents of the method before carrying out

the import.

The import/export function is multi-platform: a method exported under Mac OS can be imported under Windows and vice versa; 4D handles the conversion of characters when necessary.

4D can export and import methods in two formats:

- 4D method (extension `.c4d`): In this format, methods are exported in encoded form. The names of objects are tokenized. This format is used in particular for exchanging methods between 4D applications and plug-ins in different languages. Conversely, it is not possible to display them in a text editor.
- Text (extension `.txt`): In this format, methods are exported in text-only form. In this case, the methods are readable using a standard text editor or a source control tool.

## Project method properties

After creating a project method, you can rename it and modify its properties. Project method properties mainly concern their access and security conditions (access by users, integrated servers or services) as well as their execution mode.

The other types of methods do not have specific properties. Their properties are related to those of the objects to which they are attached.

To display the **Method Properties** dialog box for a project method, you can either:

- in the [Code Editor](#), select the **Method Properties...** command in the **Method** menu,
- or on the **Methods** page of the Explorer, **right-click** on the project method and select **Method Properties...** in the context menu or options menu.

A batch setting function allows you to modify a property for all or part of the database project methods in a single operation (see [Batch setting for method attributes](#)).

### Name

You can change the name of a project method in the **Name** area of the **Method Properties** window or in the Explorer.

The new name must comply with 4D naming rules (see [Identifiers](#)). If a method with the same name already exists, 4D displays a message saying that the method name has already been used. If necessary, 4D sorts the list of methods again.

#### CAUTION

Changing the name of a method already used in the database can invalidate any methods or formulas that use the old method name and runs the risk of disrupting application functioning. You can rename the method manually but it is strongly recommended to use the renaming function for project methods, described in [Renaming](#). With this function, you can automatically update the name wherever the method is called throughout the Design environment.

With 4D Server, the method name is changed on the server when you finish editing it. If more than one user is modifying the method name at the same time, the final method name will be the name specified by the last user to finish editing it. You may want to specify a method owner so that only certain users can change the method's name.

## INFO

Database methods cannot be renamed. The same goes for triggers, form methods, and object methods, which are bound to objects and take their names from the object concerned.

## Attributes

You can control how project methods are used and/or called in different contexts using attributes. Note that you can set attributes for an entire selection of project methods using the Explorer (see following section).

### Invisible

If you do not want users to be able to run a project method using the **Method...** command of the **Run** menu, you can make it Invisible by checking this option. An invisible method does not appear in the method execution dialog box.

When you make a project method invisible, it is still available to database programmers. It remains listed on the method list of the Explorer and of the Code Editor.

### Shared by components and host database

This attribute is used within the framework of components. When it is checked, it indicates that the method will be available to components when the application is used as the host database. On the other hand, when the application is used as a component, the method will be available to the host databases.

For more information about components, refer to the [Developing and installing 4D components](#) chapter.

### Execute on Server

This attribute is only taken into account for a 4D application in client-server mode. When this option is checked, the project method is always executed on the server, regardless of how it is called.

For more information on this option, refer to [Execute on Server attribute](#).

### Execution mode

This option allows you to declare the method eligible for execution in preemptive mode. By default, 4D executes all the project methods of your applications in cooperative mode.

If you want to benefit from the preemptive mode feature, you must explicitly declare all the methods that you want to be started in preemptive mode. The compiler will then check that these methods are actually thread-safe.

**Note:** Execution in preemptive mode is only available in compiled mode. For more information, refer to the [Preemptive 4D processes](#) section.

The following options are provided:

- **Can be run in preemptive processes:** By checking this option, you declare that the method is capable of being run in a preemptive process and therefore should be run in preemptive mode whenever possible. The "preemptive" property of the

method is set to "capable".

When this option is checked, the 4D compiler will verify that the method is actually capable and will return errors if this is not the case -- for example, if it directly or indirectly calls commands or methods that cannot be run in preemptive mode (the entire call chain is parsed but errors are only reported to the first sublevel). You can then edit the method so that it becomes thread-safe, or select another option.

If the method's preemptive capability is approved, it is tagged "thread-safe" internally and will be executed in preemptive mode whenever the required conditions are met. This property defines its eligibility for preemptive mode but does not guarantee that the method will actually be run in preemptive mode, since this execution mode requires a specific context (see [When is a process started preemptively?](#)).

- **Cannot be run in preemptive processes:** By checking this option, you declare that the method must never be run in preemptive mode, and therefore must always be run in cooperative mode, as in previous 4D versions. The "preemptive" property of the method is set to "incapable".

When this option is checked, the 4D compiler will not verify the ability of the method to run preemptively; it is automatically tagged "thread-unsafe" internally (even if it is theoretically capable). When called at runtime, this method will "contaminate" any other methods in the same thread, thus forcing this thread to be executed in cooperative mode, even if the other methods are thread-safe.

- **Indifferent**(default): By checking this option, you declare that you do not want to handle the preemptive property for the method. The "preemptive" property of the method is set to "indifferent".

When this option is checked, the 4D compiler will evaluate the preemptive capability of the method and will tag it internally as "thread-safe" or "thread-unsafe". No error related to preemptive execution is returned. If the method is evaluated as thread-safe, at runtime it will not prevent preemptive thread execution when called in a preemptive context. Conversely, if the method is evaluated "thread-unsafe", at runtime it will prevent any preemptive thread execution when called.

Note that with this option, whatever the internal thread safety evaluation, the method will always be executed in cooperative mode when called directly by 4D as the first parent method (for example through the [New process](#) command). If tagged "thread-safe" internally, it is only taken into account when called from other methods inside a call chain.

**Particular case:** If the method has also the [Shared by components and host database](#) property, setting the **Indifferent** option will automatically tag the method as thread-unsafe. If you want a shared component method to be thread-safe, you must explicitly set it to **Can be run in preemptive processes**.

## Available through

Availability attributes specify the external services which are allowed to explicitly call the method.

## Web Services

This attribute lets you publish the current method as a Web Service accessible via SOAP requests. For more information, refer to the [Publication and use of Web Services](#) chapter. When this option is checked, the **Published in WSDL** option is enabled.

In the Explorer, project methods that are offered as a Web Service are given a specific

icon .

**Note:** You cannot publish a method as a Web service if its name includes characters that do not comply with XML nomenclature (e.g. containing spaces). If the method name is not in keeping with this, 4D does not assign the property.

## Published in WSDL

This attribute is only available when the "Web Service" attribute is checked. It lets you include the current method in the WSDL of the 4D application. For more information about this, refer to [Generation of the WSDL](#).

In the Explorer, project methods that are offered as a Web Service and published in WSDL are given a specific icon .

## 4D tags and URLs (4DACTION...)

This option is used to reinforce 4D Web server security: when it is not checked, the project method cannot be executed via an HTTP request containing the special [4DACTION URL](#) used for calling 4D methods, nor the special [4DSCRIPT](#), [4DTEXT](#) and [4DHTML tags](#).

In the Explorer, project methods with this attribute are given a specific icon .

For security reasons, this option is unchecked by default. Each method that can be executed using the special Web URL or tags must be indicated individually.

## SQL

When it is checked, this option allows the project method to be executed by the SQL engine of 4D. By default, it is not selected, which means that, unless explicitly authorized, 4D project methods are protected and cannot be called by the SQL engine of 4D.

This property applies to all internal and external SQL queries --- executed via the ODBC driver, SQL code inserted between the [Begin SQL/End SQL](#) tags or the [QUERY BY SQL](#) command.

### Notes:

- Even if a method has the "SQL" attribute, access rights set at the level of the database settings and method properties are taken into account for the execution of the method.
- The ODBC **SQLProcedure** function only returns project methods with the "SQL" attribute.

For more information, refer to [4D SQL engine implementation](#) in the 4D SQL manual.

## REST Server

*This option is deprecated. Calling code through REST calls is only supported with [ORDA data model class functions](#).*

## Batch setting for method attributes

Using the "Attributes for methods" dialog box, you can modify an attribute (Invisible, Offered as a Web Service, etc.) for all or part of the database project methods in a single operation. This feature is especially useful for modifying the attributes of a large

number of project methods. It can also be used during development to apply common attributes to groups of similar methods quickly.

For batch setting of method attributes:

1. On the [Methods Page](#) of the 4D Explorer, expand the options menu, then choose the **Batch setting of attributes...** command. The **Attributes for methods** dialog appears.
2. In the "Matching method name:" area, enter a string that lets you designate the methods you want to modify as a batch. The character string is used as a search criterion for the method names.

Use the wildcard character @ to help define groups of methods:

- To designate methods whose names begin with..., type @ at the end of the string. For example: `web@`
- To designate methods whose names contain..., type @ in the middle of the string. For example: `web@write`
- To designate methods whose names end with..., type @ at the beginning of the string. For example: `@write`
- To designate all of the methods, just type @ in the area.

#### Notes:

- The search does not take upper/lower case into account.
  - You can enter several @ characters in the string, for example `dtr_o_@web@pro.@`
3. In the "Attribute to Modify" area, choose an attribute from the drop-down list, then click on the **True** or **False** radio button corresponding to the value to be applied.

**Note:** If the "Published in WSDL" attribute is set to True, it will only be applied to project methods already containing the "Offered as a Web Service" attribute.

4. Click on **Apply**. The modification is applied instantly to all the project methods designated by the character string entered.

## Code Editor

4D has a powerful built-in code editor that offers a wide set of features for highly productive code editing such as intelligent code completion, code navigation, debugging, searching, and more.

The Code Editor works much like a text editor. Writing a method or a class is usually a combination of typing text, selecting components, and dragging items from the Explorer or other windows. You can also use various type-ahead functions to create methods faster.

You can scroll through the contents of methods, classes and functions, which can include up to 32,000 lines of code or 2 GB of text.

The 4D Code Editor provides basic syntax error-checking. Additional error-checking is performed when the code executes. For more information on how to handle errors, see [Debugging](#).

## Interface

### Toolbar

Each Code Editor window has a toolbar that provides instant access to basic functions related to code execution and editing.

| Element                   | Icon | Description   |
|---------------------------|------|---|
| Method execution          |      | <p>When working with methods, each Code Editor window has a button that can be used to run the current method. Using the menu associated with this button, you can choose the type of execution:</p> <ul style="list-style-type: none"> <li>• <b>Run new process:</b> Creates a process and runs the method in standard mode in this process.</li> <li>• <b>Run and debug new process:</b> Creates a new process and displays the method in the Debugger window for step by step execution in this process.</li> <li>• <b>Run in Application process:</b> Runs the method in standard mode in the context of the Application process (in other words, the record display window).</li> <li>• <b>Run and debug in Application process:</b> Displays the method in the Debugger window for step by step execution in the context of the Application process (in other words, the record display window).</li> </ul> <p>For more information on method execution, see <a href="#">Calling Project Methods</a>.</p> |
| Find in method            |      | Displays the <a href="#">Search area</a> .  |
| Macros                    |      | Inserts a macro at the selection. Click the dropdown arrow to display a list of available macros. For more information on how to create and instantiate macros, see <a href="#">Macros</a> .  |
| Expand all / Collapse all |      | These buttons allow expanding or collapsing all the control flow structures of the code.  |
| Method information        |      | Displays the <a href="#">Method Properties</a> dialog box (project methods only).   |
| Last clipboard values     |      | Displays the last values stored in the clipboard.   |
| Clipboards                |      | Nine clipboards available in the code editor. You can <a href="#">use these clipboards</a> by clicking on them directly or by using keyboard shortcuts. You can use a <a href="#">Preferences option</a> to hide them.  |
| Navigation dropdown       |      | Lets you navigate inside methods and classes with automatically tagged content or manually declared markers. See below  |

## Editing area

This is where you write and edit your code. The editor automatically indents code text and colors the different syntax elements for clear code structure.

You can customize the display of the editing area. Any customization is automatically passed on to all the windows of the code editor:

| Option                                    | Description  | Set in...  |
|---|--|--|
| <b>font and font size</b>                 | Sets the character font and size to be used in the editing area  | <b>Preferences &gt; Methods</b><br>or <b>Method &gt; View &gt; Bigger Font or Smaller Font</b>                         |
| <b>style and color of syntax elements</b> | assign a specific color and/or style to each type of element of the 4D language. You can also change the different colors used in the interface of the editing area (highlighting, background, and so on). | Right-click on a language element (variable, keyword, etc.) > <b>Style</b> submenu. Or <b>Preferences &gt; Methods</b> |
| <b>spaces</b>                             | You can display the spaces between words using dots (.) instead of blank spaces. This option applies to all the code elements (command names, variables, comments, etc.).                                  | <b>Method &gt; View &gt; White Spaces</b>  |
| <b>themes</b>                             | You can select the Dark or Light theme, or set a custom one  | <b>Preferences &gt; Methods</b>  |
| <b>width of code indentations</b>         | Set the width of code indentations   | <b>Preferences &gt; Methods</b>  |

## Change bars

Colored bars instantly show you where lines of code were modified since the method was opened:

```
14 ALL RECORDS ([DOCUMENTATIONS])
15 ARRAY TEXT (<>KeyList;0)
16 ARRAY TEXT (<>PluralKeyList;0)
```

The change bars change colors to indicate whether or not the modifications were saved:

- yellow: Line was modified and method has not yet been saved.
- green: Line was modified and method has been saved.

## Lists area

The lists area lets you display one or more lists of elements necessary for writing methods and classes (commands, constants, forms, etc.). You can choose the number and contents of the lists displayed in the window.

By default, the Code Editor displays four lists. You can hide or show all lists by clicking on the icon at the bottom right of the window. 

You can enlarge or reduce the relative width of each list area by dragging one of its partitions. It is also possible to adjust the size of the list area in relation to that of the editing area by dragging the dividing line between them.

- Double-clicking on an item in a list causes it to be inserted into the editing area, at the location of the cursor.
- To **modify the contents** of a list, click on the title area of the list concerned: a pop-up menu appears, enabling you to choose the type of item to be displayed.



- To add or remove a list, click in the title area of one of the lists and choose the corresponding command in the pop-up menu. The **Remove this list** command is disabled when you click on the last list. If you want to hide all the lists, you must either click on the **show or hide lists** button at the bottom right of the window or hide them by default in the **Preferences**.
- You can hide the lists in all the windows in the following ways:
  - Select the **View > Lists** option in the **Method** menu (a check mark indicates whether lists are displayed)
  - Uncheck the **Preferences > Methods > Options > Show Lists** option. For the modifications made in the **Preferences** dialog box to be taken into account, any open methods, classes or functions must first be closed then reopened.

## Available lists of items

You can display the following lists of items in the lists area of the Code Editor window:

- **All tables and fields:** Database table and field names in the form of a hierarchical list. When you insert a field name into the method by double-clicking on its name, 4D inserts it while respecting the syntax and adds the name of the table or subtable as the case may be.
- **Table** (submenu): Field names of the table selected using the submenu.
- **Current table:** Field names of the current table (available in triggers, form methods and object methods).
- **Project forms:** Database project form names. When you double-click on a project form name, 4D inserts its while respecting the syntax: the form name is inserted between quotes.
- **Table forms:** Database table and form names in the form of a hierarchical list. When you insert a form name into a method by double-clicking its name, 4D inserts it while respecting the syntax: the form name is inserted between quotes and is preceded by the name of the table and a semi-colon. For example: [Table];"Form".
- **Methods:** Database project method names.
- **All folders:** Names of object folders and subfolders set in the database displayed in the form of a hierarchical list. Folders can be used to organize objects in a customized manner. They are managed from the Home Page of the Explorer.

- **Folders** (submenu): Contents of the folder selected using the submenu.
- **Macros**: Macro names defined for the database (see [Creating and using macros](#)).
- **Commands**: 4D language commands in alphabetical order.
- **Commands by themes**: 4D language commands classified by theme in the form of a hierarchical list.
- **Menu bars**: Names and numbers of menu bars [created with the 4D Menu bar editor](#).
- **Constants**: 4D constants and those of any plug-ins, classified by theme in the form of a hierarchical list.
- **Lists**: Names of lists.
- **All plug-in commands**: Commands for all the plug-ins installed in the database (if any), classified by theme in the form of a hierarchical list.
- **SQL Keywords**: set of keywords recognized by the 4D SQL syntax parser. This list includes commands (e.g. SELECT), clauses (e.g. WHERE) as well as functions (ABS).
- **SQL Functions**: 4D SQL functions.

**Note:** Except for the Macros element, all the lists are in alphabetical order.

## Save as template

You can save the lists set in the Code Editor window in the form of a template. Once the template is saved, the parameters set in it will be used for each new Code Editor window that is opened.

The following parameters are stored in the template:

- Relative size of the editing and list areas
- Number of lists
- Location and contents of each list
- Relative width of each list

To save a Code Editor window as a template, choose **Method > Save As Template**.

The template is saved immediately (no dialog box appears). It is stored in the **Preferences** of the 4D application. If a previous template already exists, it is replaced.

## Break points area

This area, located to the left of the editing area, allows you to display the line numbers and to insert break points directly next to specific instructions. Break points are useful during the debugging phase of your programming. They stop the execution of your code at specific locations and display the debugger.

For more information on break points, see the [Debugging](#) section.

You can display or hide the line numbers in the break points area for each window of the Code Editor.

- To enable or disable the display of line numbers by default, choose **Preferences > Methods > Show line numbers**.
- To modify this display separately for each window of the Code Editor, choose **Method > View > Line Numbers**.

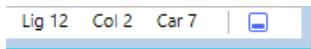
Displaying the line numbers makes it easier to find your way around in the window. The **Method > Go to Line Number...** command in the also lets you take advantage of this display.

This type of search is useful when used in conjunction with the [compiler](#), which flags

runtime errors by the line number in which they occur.

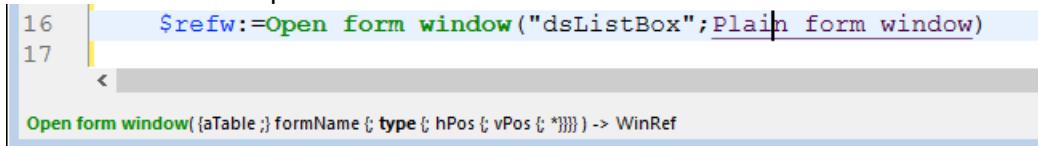
## Status bar

The status bar located at the bottom right part of the editor window displays the position of the cursor at all times:



- **Ln:** Line number
- **Col:** Column number, i.e., the level in the hierarchy of programming structures. The first level is 0. The column number is useful for debugging since this information can be provided by the interpreter in the event of an error in the code.
- **Ch:** Location of character in the line.
- Hide/display lists.

When you set the cursor in a command, function or parameter(s), the status bar displays the syntax of the command. If you write or select a parameter, the area shows the current parameter in **bold**:



## Navigation dropdown

The navigation dropdown helps you organize your code and navigate more easily inside your classes and methods:

Some tags are added automatically, and you can complement the dropdown list using [markers](#).

## Code navigation

Click an item in the dropdown list to go to its first line in the code. You can also navigate with arrow-keys and press **Enter**.

## Automatic tagging

Constructors, method declarations, functions and computed attributes are automatically tagged and added to the dropdown list.

When there is no tag in the class/method, the tool displays "No tag".

The following items are added automatically:

| Icon | Item   |
|------|--|
|      | No tag   |
|      | Class constructor or method declaration          |
|      | Computed attribute (get, set, orderBy and query) |
|      | Class function name                              |

## Manual tagging

By adding markers in your code, you can add the following tags to the dropdown:

| Icon | Item       |
|------|------------|
|      | MARK: tag  |
|      | TODO: tag  |
|      | FIXME: tag |

You declare them by adding comments such as:

```
// FIXME: Fix following items
```

Declarations are not case-sensitive; writing `fixme:` has the same effect.

Adding a hyphen after the `MARK:` tag draws a separating line in the code editor and the dropdown menu. So writing this:

```
// FIXME: Fix following lines

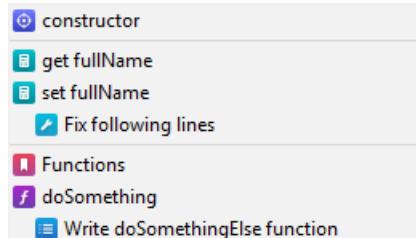
This.firstName:=Substring($fullName; 1; $p-1)
This.lastName:=Substring($fullName; $p+1)

//MARK:- Functions I

Function doSomething

// TODO: Write doSomethingElse function
```

Results in this:



All markers located inside functions are indented in the dropdown list, except for the `MARK:` tags located at the end of functions and not followed by instructions. Those will appear at the first level.

## Display order

Tags are displayed in their appearing order inside the method/class.

To display the tags of a method or class in alphabetical order, do one of the following:

- **right-click** the dropdown tool
- hold **Cmd** on macOS or **Alt** on Windows, and click the dropdown tool

Tags inside functions move with their parent items.

## Shortcuts

Multiple features of 4D's Code Editor are available through default keyboard shortcuts.

MACOS

Under macOS, use the **Command** key instead of the **Ctrl** key mentioned (Windows).

| <b>Shortcut</b>                 | <b>Action</b>  |
|---------------------------------|--|
| <b>Selection and navigation</b> |  |
| Double-click                    | Select a language element name   |
| [Alt]+Double-click              | Select a language element name containing spaces (constant, method, etc.)  |
| [Shift]+[right arrow]           | Create and enlarge the selection, character by character, to the right, or Reduce the selection, character by character, from the left |
| [Shift]+[left arrow]            | Reduce the selection, character by character, from the right or Create and enlarge the selection, character by character, to the left  |
| [Shift]+[down arrow]            | Create and enlarge a selection, line by line, from the top to the bottom   |
| [Shift]+[up arrow]              | Create and enlarge a selection, line by line, from the bottom to the top   |
| [Ctrl]+[Shift]+[right arrow]    | Create and enlarge the selection, word by word, from the right   |
| [Ctrl]+[Shift]+[left arrow]     | Reduce the selection, word for word, from the right, or create and enlarge the selection, word by word, from the left                  |
| [Ctrl]+[right arrow]            | Move the insertion point, word by word, from left to right   |
| [Ctrl]+[left arrow]             | Move the insertion point, word by word, from right to left   |
| [Alt]+[down arrow]              | Move the line(s) where the cursor is to the bottom   |
| [Alt]+[up arrow]                | Move the line(s) where the cursor is to the top  |
| [Home]                          | Place the insertion point at the beginning of the line   |
| [End]                           | Place the insertion point at the end of the line   |
| [Ctrl]+[Home]                   | Place the insertion point at the beginning of the method   |
| [Ctrl]+[End]                    | Place the insertion point at the end of the method   |
| [Shift]+[Home]                  | Select all the characters in the line that are to the left of the cursor   |
| [Shift]+[End]                   | Select all the characters in the line that are to the right of the cursor  |
| [PgUp]                          | Scroll the contents of the method, page by page, from the bottom to the top (doesn't modify the insertion point)                       |
| [PgDn]                          | Scroll the contents of the method, page by page, from the top to the bottom (doesn't modify the insertion point)                       |

### **Introspection**

|   |  |
|---|--|
| [Ctrl]+K or [Alt]+double-click                          | Same as <a href="#">Goto definition</a> command  |
| [Ctrl] (Windows) or [Alt] (macOS)+hovering over a token | Underline the token (identified language element). Click on the underlined token = same as <a href="#">Goto definition</a> command |

### **Find and replace**

|                        |                    |
|------------------------|--------------------|
| [Ctrl]+F               | Find               |
| [Ctrl]+G               | Find Next          |
| [Ctrl]+[Shift]+G       | Find Previous      |
| [Ctrl]+E               | Find Same Next     |
| [Ctrl]+[Shift]+E       | Find Same Previous |
| [Ctrl]+[Alt]+F         | Replace            |
| [Ctrl]+[Alt]+G         | Replace Next       |
| [Ctrl]+[Alt]+[Shift]+G | Replace Previous   |

### **Clipboards**

| <b>Shortcut</b>   | <b>Action</b>   |
|---|---|
| [Shift]+click [Alt]+click on clipboard icon   | Copy selected text to a clipboard                         |
| [Ctrl]+[Shift]+number key   | Copy selected text to the number clipboard                |
| [Ctrl]+click on clipboard icon  | Paste contents of a clipboard at cursor location          |
| [Ctrl]+number key   | Paste contents of the number clipboard at cursor location |
|  TIP |   |

Most of these shortcuts can be customized in the [4D Preferences](#) dialog box.

## Editing Code

4D uses standard text editing techniques for typing and editing in the Code Editor.

The Code Editor uses display conventions (style, color) for the syntax elements. You can [customize these conventions](#). As you type, when you validate your entry, 4D evaluates the text of the line and applies the appropriate display format. 4D also indents each line to its proper level in relation to the preceding line when you use programming structures (If, End if...).

You can use the arrow keys to move from line to line quickly. Using the arrow keys to move across several lines is quicker than clicking because the editor delays evaluating the line for errors.

Under Windows, the code editor includes an Input Code Editor (IME) to facilitate code editing on Japanese or Chinese systems.

The Code Editor includes numerous [navigation shortcuts](#).

## Using the backslash

The backslash character (\) has a specific support in the 4D language:

- inserted at the end of lines, it allows to write a single statement on [several lines](#).
- it allows to define [escape sequences](#).



### CAUTION

The backslash character (\) is used as a separator in [pathnames under Windows](#). In general, 4D will correctly interpret Windows pathnames entered in the Code Editor by replacing the single backslash \ with a double backslash \\. For instance,

C:\MyDocuments will become C:\\MyDocuments. However, if you write

"C:\MyDocuments\New", 4D will display "C:\\MyDocuments\\New". In this case, the second backslash is interpreted incorrectly as \\N (an existing [escape sequence](#)). You must therefore enter a double backslash \\ when you want to have a backslash in front of a character used in one of the escape sequences recognized by 4D.

## Drag-and-drop

From the Explorer, you can drag and drop tables, fields, forms, project methods, constants, or 4D commands. When you drag and drop an element, 4D always uses the correct syntax. For example, if you drag the field name First Name from the [People] table, it appears in the Code Editor as [People]First Name. Similarly, if you drag the Form name Input from the People table, it appears in the Code Editor as [People];"Input".

When you insert a command by dragging it from the **Commands** page of the Explorer, it appears with its syntax (which consists of all of its parameters) in the Code Editor. This feature simply reminds you of the parameters that the command expects. You can then use a syntax that better suits your usage.

You can also drag-and-drop within a method, class, function or between two different ones. In the Code Editor, the drag-and-drop mechanism is activated as soon as a portion of text is selected. By default, the drag-and-drop mechanism **moves** the selected text. In order to **copy** it, hold down the **Ctrl** key (Windows) or the **Option** key (macOS) during the operation.

## Changing case

You can automatically modify the case of selected characters using the commands in **Methods > Case** or the context menu of the editor:

- **Uppercase / Lowercase**: Switch the selected characters to uppercase or lowercase.
- **camelCase / CamelCase** : Switch the selected characters to "camel case". This consists in changing each first letter of a group of attached words to uppercase. This type of notation is often used for variable nomenclatures. hireDate and PurchaseDate are examples of two variants of camel case notation.

When you apply one of these commands to a text selection, the spaces and "\_" characters are removed and the first letter of each word becomes uppercase.

## Swap expression

The **Swap Expression** function can be used to reverse the arguments of an expression assigning a value. For instance,

```
variable1:=variable2
```

becomes

```
variable2:=variable1
```

This function is extremely useful for reversing a set of assignments used to get or set properties, or to correct input errors. To use this function, select the line(s) to be modified, then choose **Method > Swap Expression** or use the context menu of the area. Within the selection, only the lines assigning a value will be modified.

## Clipboards

In addition to the standard copy-paste operation, 4D offers two additional functions that let you work with the contents of different clipboards:

- The program stores the last 10 "copy" or "cut" actions that were performed in the Code Editor in memory during the current session. Each of the different contents saved in this way can be reused at any time. To do this, use the **Clipboard History** command of the Code Editor context menu or the "Last Clipboard values" button of the toolbar:

```

SET ABOUT(Get indexed str...
If ($OldLang#<>StrLang)\r...
CLOSE WINDOW($Win)\r
M_Contacts
$OldLang
<>PS_About
<>StrLang
ALERT("The operation has ...
//CONFIRM("WARNING: If yo...
//CONFIRM("Voulez-vous aj...

```

The first few words of the copied or cut items are displayed. Select an item to insert it at the current location of the cursor.

- Nine additional numbered clipboards are available and can be employed directly using the buttons of the Code Editor toolbar or [using keyboard shortcuts](#):



## Moving lines

You can move the line where the cursor is directly without selecting it first using the **Move Lines Up** and **Move Lines Down** commands in the **Method** menu. You can also do this using the **Alt/Option + Up Arrow** or **Down Arrow** [keyboard shortcut](#).

## Autocomplete functions

The Code Editor provides autocomplete functions. 4D automatically displays suggestions based on the first few characters typed.

In the example given below, typing the string "cop" causes the display of a blue triangle indicating that several suggestions are available:

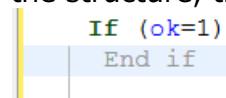


When the characters you enter correspond to a single possibility, this suggested value appears greyed out (and is inserted if you hit the **Tab** key): **ale|RT** ---> **ALERT|**

If you checked the **Insert () and closing } ) ] "** option in the **Methods** page of the **Preferences**, 4D will also automatically add **()** after a 4D command, keyword or project method that requires one or more mandatory arguments (after accepting a suggestion or completion): **ale|RT ()** ->

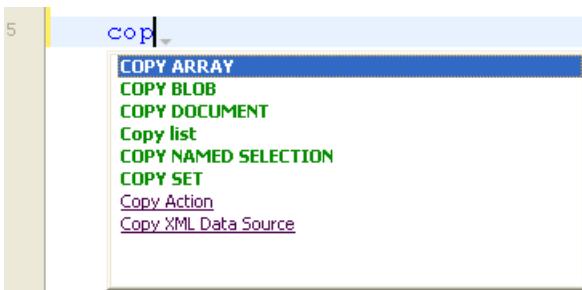


Autocompletion also works with code structures (e.g. If..End if, For each...End for each): when you enter the first part of the structure, the Code Editor will



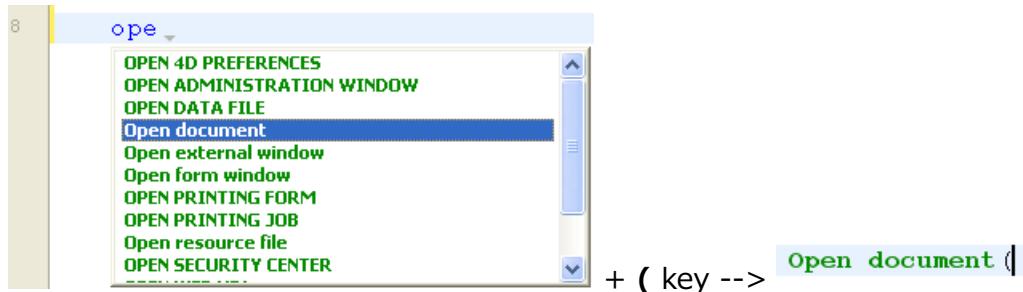
automatically suggest the closing part:

If there are several suggestions available, 4D displays them in a pop-up list when you hit the **Tab** key:



The list is in alphabetical order. Choose the value by double-clicking it or scroll the list using the arrow keys of the keyboard and then hit **Enter**, **Carriage Return** or **Tab** to insert the selected value.

By default, you can also insert a suggested value by hitting one of the following delimiter `( ; : = < [ {` keys after selecting a value: the value inserted is then followed by the delimiter, ready for data entry.



You can disable the use of delimiters for inserting suggested values in **Preferences > Methods > Options**.

You can press the **Esc** key to close the pop-up list or you can continue typing while it is open. The values suggested in the pop-up list are updated as additional characters are typed.

If the characters typed correspond to different types of objects, the list displays them in their current style. The following types of objects can be displayed:

- 4D commands
- SQL commands
- User methods
- Table names
- Field names
- Constants
- Local, process or interprocess variable, declared in the method
- Object property names
- Plug-in commands
- 4D keywords
- SQL keywords
- Macros (displayed between `< >`)

For practical reasons, you can disable the automatic display of the list of suggestions for **constants**, **(local or interprocess) variables** and **object attributes** and/or **tables**. These options are found in **Preferences > Methods > Options**

## Object property names

4D automatically displays case-sensitive suggestions of all valid object property names in 4D code when you:

- type a dot "." after an object or
- use the Tab key after a dereferenced object pointer "->".

The `length` property is always included for use with collections.

Once created, property names are stored in an internal global list and are available anytime a method/class/function is opened, closed or changes focus.

The list of suggestions is dynamically updated while you edit code. When switching between windows, new/edited property names are always added to the global list. The list is also updated when you preview a method, class or function in the Explorer.

When the database is restarted, the list is reinitialized.

You can disable the automatic display of object properties in **Preferences > Methods > suggestions**.

## Find and replace

The Code editor has powerful **find and replace** features that apply to the current window.

A search and replace area can be displayed in the toolbar of any method window:

To display this area, click on the **Find in method** icon of the [toolbar](#) or select a find or replace function either through a [shortcut](#) or a command from the **Edit > Find** submenu. You can close this area at any moment by clicking on the **x** button at the rightmost side of the toolbar.



The **Find in Design** feature in the 4D toolbar or in the **Edit** menu is not specific to the Code editor but may be used to search for a value among all the methods and classes.

### Find

Select **Find > Find...** in the **Edit** menu or type **Ctrl+F** (Windows)/**Cmd+F** (macOS) to display/enable the *Search* area.

The search defined in this area will be performed in the code located in the window.

The **find** entry area enables you to enter the string to be searched for. This area is a combo box that stores the last 10 strings that have been searched for or replaced during the session. If you highlight text before choosing the **Find...** command, it will appear in this area. You can then either use this text or replace it with another.

Once a string is entered or selected, all occurrences found in the opened window are highlighted and the right side of the area displays the total number of hits found. It also indicates the current position of the cursor among all hits.

Hit the **Enter** key to select closest occurrence to the cursor. You can also click on the **Next / Previous** buttons to select all occurrences sequentially towards the beginning or end of the current method, starting from the initial location of the cursor,

or use the **Find Next** and **Find Previous** commands of the [Edit menu](#).

## Options

- **Case Sensitive** : Take the case of characters as they were entered in the find area into account. This option also takes into account diacritic characters. For instance, a search for "MyVar" will not find "myVar"; a search for "dej" will not find "déjà".
- **Whole Word** : Limit the search to exact occurrences of the word being searched for. When this option is checked, for instance, a search for "client" will not find either "clients" or "myclient." By default, this option is not checked; therefore, a search for "var" will find "Myvar," "variation," etc.

## Replace

Click on the **v** toggle button on the left side of the *Search* area to display/hide the *Replace* area. You can also select **Find > Replace...** in the **Edit** menu or type **Ctrl+Alt+F** (Windows)/**Cmd+Alt+F** (macOS) .

The *Replace* entry area is used to define the character string that will replace the one defined above.

Click the **Replace** button to launch the search with all defined options and replace the first occurrence found. 4D begins searching from the current text insertion point and continues to the end of the method. It is then possible to continue finding/replacing using the **Replace Next** and **Replace Previous** commands of the [Edit menu](#).

Click the **Replace all** button to replace all the occurrences corresponding to the search criteria directly in the open method.

## Find Same

The **Find Same** command is used to find character strings identical to the one selected. This command is only active if you have selected at least one character in the Code Editor.

The search carried out is of the "Find Next" type in the current code editor window.

The **Find Same Next** and **Find Same Previous** commands are used to find character strings *strictly* identical to the ones selected. For example, the case must match.

## Bookmark All

The **Edit > Bookmark All** command is enabled when a search has already been specified in the find or replace dialog box. When you select this command, 4D puts a bookmark at each line that contains an item corresponding to the "current" search criteria. This makes it easy to spot all the search results. For more information about bookmarks, refer to [Bookmarks](#).

## Syntax errors

4D automatically checks the method syntax to see if it is correct. If you enter text or select a component that is not syntactically correct, 4D displays a symbol to indicate the incorrect expression . When you move the mouse over the symbol, a help tip displays the cause of the error:



When entering code, you can immediately check the syntax of the current line (without advancing to the next line) by pressing the **Enter** key on the numeric keypad. 4D evaluates the line, formats it, marks any errors, and places the insertion point at the end of the line. When a line of a method, class or function is marked as having improper syntax, check and fix the entry. If the line becomes correct, 4D removes the error symbol. When you save or close the window, the entire method is validated. You can also force validation by pressing the **Enter** key.

When the method, class or function is validated, 4D checks for:

- basic syntax errors
- the structure of statements (`If`, `End if` and so on)
- matching enclosing characters in the code such as parentheses or quotation marks. When you type an enclosing character, 4D indicates the match by framing the start/end characters with gray rectangles:

```
ALERT ("Pi is equal to:" +String(Arctan(1)*4))
```

If you click on an enclosing character in the code, 4D indicates its match with gray rectangles by default. You can modify the way 4D indicates matching enclosing characters or disable this feature in **Preferences > Methods > Options > Matching parentheses**.

The Code Editor can only check for obvious syntax errors (misspellings and the like). It does not check for errors that only occur during execution. Execution errors are caught by 4D when the code is executed.

4D has a built-in debugger (see [Debugging](#)) for handling and correcting these errors. The compiler also provides indispensable help for detecting errors. For more information about the compiler, refer to the [Compilation](#) chapter.

## Help tips

The Code Editor provides various contextual information using help tips. They appear when you mouse over an object.



The [status bar](#) also provides contextual information.

- **Errors:** When you mouse over the symbol indicating an error to the left of the editing area, a help tip displays the cause of the error (see [Syntax errors](#)).
- **4D command documentation:** When you move the mouse over a 4D command or function, a help tip provides its syntax along with a brief description of how it

`CLEAR SET ("Current Book Only")`  
works.  
`CLEAR SET (set)`  
clears set from memory and frees the memory used by set.

- **Variable type and description:** When you mouse over a variable, a help tip shows its type (if it has been explicitly defined in the method) and associated

`C_OBJECT ($person) /*prop1:age, prop2:name*/`  
`$person:=New object`  
`$person : C_OBJECT`  
`/*prop1:age, prop2:name*/`  
comment, if any.

- **Project methods or functions:** When you mouse over a project method or class

function, a help tip displays:

- either the comments specified in the Explorer.
- or the first few lines of the method or class function if it includes comments (lines beginning with // or /.../ comment block). It is common practice to insert documentation for the method as well as its parameters in the form of comments at the beginning of the method. You can get this information directly in the help tip, just make sure to first remove any comments found in the Explorer. Comments at the beginning of a method:

Help tip in another method:

- You can also create a **dedicated documentation file** named <MethodName>.md in the <package>/documentation folder. See [Viewing documentation in the code editor](#)

## Comment / Uncomment

The 4D language supports [comments](#), which are inactive lines of code. The code editor does not apply any particular style within comments. The length of comments is limited to the maximum size of 32,000 characters per line. There is no limit on the number of lines.

There are two kinds of comments: //comment (single line comment) and /\*comment\*/(inline comments or multiline comment blocks).

Comments can be created by typing / characters. Or, the **Comment/Uncomment** command, found in the **Method** menu as well as in the Code Editor context menu is used to mark a group of selected lines of code as single line comments, or, on the contrary, to remove the single line comment characters from a selection. To use this command, select the code to be marked as commented, then select the **Comment/Uncomment** command:

```
12  IF (Count parameters>=2)
13    EXECUTE FORMULA(Command name (55) + " ([ "
14  ELSE
15    IF (Count parameters>=1)
16      FORM SET INPUT ($1->;"INPUT FORM")
17      End if
18    End if
-->
12  IF (Count parameters>=2)
13    EXECUTE FORMULA(Command name (55) + " ([ "
14    //Else
15    //If (Count parameters>=1)
16    //FORM SET INPUT ($1->;"INPUT FORM")
17    //End if
18  End if
```

When the selection contains only active code, the **Comment** command is applied. When the selection includes both active code and commented lines, an additional pair of comment characters ( // ) is added to the latter; this way, they will retain their initial commented status if the line is subsequently "uncommented." When the selection contains only commented lines, the **Uncomment** command is applied.

The **Comment/Uncomment** command only operates with full lines --- it cannot be used to comment only part of a line.

## Expand / Collapse

4D code located inside loops and conditions can now be collapsed or expanded, in order to facilitate the reading of methods:

- Expanded code:

```
12  IF (Count parameters>=2)
13    EXECUTE FORMULA(Command name(55)+"(+"+$2+"];"INPUT FORM")")
14  ELSE
15    IF (Count parameters>=1)
16      FORM SET INPUT ($1->;"INPUT FORM")
17    End if
18  End if
19
```

- Collapsed code:

```
12  IF (Count parameters>=2)
13    EXECUTE FORMULA(Command name(55)+"(+"+$2+"];"INPUT FORM")")
14  ELSE
15    +IF (Count parameters>=1) ...
16    End if
19
```

If you place the mouse over the expand button [...], a help tip appears, displaying the first lines of the hidden code.

A collapsed portion of code can be selected, copied, pasted or deleted. All the lines included therein will be copied, pasted or deleted respectively. When a portion of code is pasted, it is automatically expanded.

There are several ways to expand and collapse code:

- Click on the expand/collapse icons ([+] and [-] under Windows) or on the opening button [...]
- Use the commands of the **Method > Collapse/Expand** submenu:
  - **Collapse Selection / Expand Selection**: collapses or expands all the code structures found in the text selection.
  - **Collapse Current Level / Expand Current Level**: collapses or expands the code structure at the level where the cursor is located. These commands are also available in the **context menu** of the editor.
  - **Collapse All / Expand All**: collapses or expands all the loops and conditions of a method. These commands are also available in the toolbar of the editor.

## Blocks

Blocks can be defined by:

- Quotation marks
- Parentheses
- A logical structure (If/Else/End if, While/End while, Repeat/Until Case of/End case)
- Braces

### Select Enclosing Block

The **Select Enclosing Block** function is used to select the "enclosing block" of the code containing the insertion point.

If a block of text is already selected, the function selects the enclosing block of the next highest level and so on, until the entire method is selected.

Pressing **Ctrl+Shift+B** (Windows) or **Command+Shift+B** (macOS) enables you to reverse this operation and deselect the last enclosing block selected.

**Note:** If the insertion point is placed in an `If` or `Else` type structure, the enclosing block will be the one containing, respectively, the `If` or `Else` statement.

## Start of Block or End of Block

Two commands make it easier to move around within code structures (e.g.

`If...Else...End if`):

- **Start Of Block:** places the cursor at the start of the current structure, just before the initial keyword.
- **End Of Block:** places the cursor at the end of the current structure, just after the final keyword.

These commands are found in the **Method** menu as well as the context menu of the editor. You can also use the following shortcuts:

- Windows: **Ctrl + up arrow** or **Ctrl + down arrow**,
- macOS: **Command + up arrow** or **Command +down arrow**.

## Bookmarks

4D lets you associate bookmarks with certain lines in your methods. You can then browse quickly within the code by passing from one bookmark to another using specific commands.



A bookmark moves along with its original row if additional rows are inserted in the method. Bookmarks are saved with the methods.

Bookmarks are managed using the **Bookmarks** submenu of the **Method** menu:

- **Toggle:** Associates a bookmark with the current line (where the cursor is located) if it does not already have one or removes the existing bookmark if it does. This function is also available using the **Toggle Bookmark** command of the editor's context menu or using the **Ctrl+F3** (Windows) or **Command+F3** (macOS) keyboard shortcut.
- **Remove All:** Removes all bookmarks from the foreground window.
- **Goto Next / Goto Previous:** Enables browsing among bookmarks in the window. Selecting one of these commands places the cursor on the first character of the line associated with the bookmark concerned. You can also use the keyboard shortcuts **F3** (go to next) or **Shift+F3** (go to previous).

### ⓘ INFO

You can use bookmarks as markers for lines that contain an [item found by a search](#). In this case, 4D automatically adds the bookmarks. For more information, refer to [Bookmark all](#).

## Reveal in Explorer

The **Reveal in Explorer...** command opens an Explorer window with the target element selected. To do this, place the cursor inside the element's name or select it, then choose **Method > Reveal in Explorer...** .

## Search Callers

The **Search Callers** command in the **Method** menu is only enabled for project methods. It searches for all the objects (other methods or menus) that reference the project method.

**Note:** The **Search Callers...** command is also available in **Explorer > Methods**

This command displays its results in a new window.

## Goto Definition

The **Goto Definition** command opens the definition of an element referenced in the Code Editor. To do this, place the cursor inside the object name or select it, and choose **Method > Goto Definition...** or use the context menu of the editor.



This feature is also available through the keyboard shortcut **Ctrl+K** (Windows) / **Command+K** (macOS) or **Alt+double-click**.

The effect of the **Goto Definition...** command varies depending on the target element:

- with a project method, it displays the contents of the method in a new window of the Code Editor
- with a class name or class function, it opens the class in the the Code Editor
- with a built-in 4D command or function, it has the same effect as the [Show documentation](#) command.

## Show documentation

The **Show documentation...** command opens the documentation for the target element. To do this, place the cursor inside the element's name or select it, then choose **Method > Show documentation...** or use the contextual menu. The effect varies depending on the target element. For example:

- Selecting a project method or a user class and choosing **Show documentation...** selects the method in the Explorer and switches to the documentation tab
- Selecting a 4D command, function, or class name and choosing **Show documentation...** displays the online documentation.
- If no element is selected, the command opens the documentation of the method currently opened in the Code Editor, [if any](#).

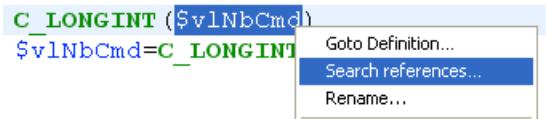


To display the documentation of a 4D "classic" language command, select the command name or simply place the cursor in the name and press **F1**. The documentation of the command is displayed in a new window of your default browser. 4D looks for the documentation depending on the settings made in the Preferences (see [Documentation location](#)).

## Search References

The **Search References...** command found in the **Method** menu or the context menu of the Code Editor finds all the objects (methods and forms) in the project where the current item of the method is referenced (used).

The current item is either the one selected or the one where the cursor is located. It can be a field name, variable name, command, string, and so on. For example, the following action looks for all the occurrences of the `vNbCmd` variable in the database:



This command displays its results in a new window.

## Creating and using macros

You can use macro-commands in your methods. Using macro-commands saves a lot of time during code entry.

### What is a macro?

A macro-command is a section of 4D code that is permanently accessible and that can be inserted anywhere in your methods, whatever the type of database open. Macros can contain all types of 4D text, commands and constants, as well as special tags which are replaced at the time of macro insertion by values derived from the method context. For instance, a macro may contain the tag `<method_name/>`; at the time of macro insertion, this tag will be replaced by the name of the current project method.

Macros are stored in one or more XML format (text) file(s). They can be placed in a Code Editor list; they can also be called using the context menu of the editor or using the autocomplete function.

4D macros are written in XML format. You can use the 4D default macro file as is or modify it.

### Location of macros

4D loads the macros from a folder named **Macros v2**. Macros must be in the form of one or more XML files that are placed in this folder.

The "Macros v2" folder can be located:

- In the active 4D folder of the machine. Macros are then shared for all the databases. **Note:** The location of the active 4D folder varies according to the operating system used. For more information, refer to the description of the [Get 4D folder](#) command in the *4D Language Reference* manual.
- Next to the database structure file. Macros are only loaded for this structure.
- For components: in the **Components** folder of the database. Macros are then only loaded if the component is installed.

These three locations can be used simultaneously: it is possible to install a "Macros v2" folder in each location. The macros will be loaded in the following order: 4D folder, structure file, component 1... component X.

### Default macros

4D offers a set of default macros containing, for example, control flow keywords. These macros are included in the default "Macros.xml" file, placed in the "Macros v2" folder that is created in the active 4D folder of the machine during the initial startup of 4D.

You can modify this file or the contents of the folder subsequently as desired (see the following paragraph). In the event of problems with this folder, it can be deleted and 4D will re-create it on the next startup.

### Adding customized macros

You can add customized macros in the "Macros.xml" file using a standard text editor or by programming. You can also add XML files of customized macros in this folder.

In local mode, the macros file can be open while using 4D. The list of available macros is updated on each event activating 4D. For instance, it is possible to bring the text editor to the foreground, modify the macro file, then return to the method: the new macro is then available in the Code Editor.

Empty or erroneous macros are not displayed.

## Checking the syntax of customized macros

The macro-command files of 4D must be in conformity with the XML standard. This means more particularly that XML declaration `<?xml version="1.0" ...?>` and document declaration `<!DOCTYPE macros SYSTEM "http://www.4d.com/dtd/2007/macros.dtd">` statements are mandatory at the beginning of a macro file in order for it to be loaded. The different types of XML encoding are supported. However, it is recommended to use encoding that is Mac/PC (UTF-8) compatible. 4D provides a DTD that can be used to validate the macro files. This file is found in the following location:

- Windows: ¥Resources¥DTD¥macros.dtd
- Mac OS: :Contents:Resources:DTD:macros.dtd

If a macros file does not contain the declaration statements or cannot be validated, it is not loaded.

## Syntax of 4D macros

4D macros are built using customized XML tags called "elements."

Some tags indicate the start and end of the definition (double tags of the type `<tag></tag>`), others are replaced by insertion context values (`<tag/>`).

In conformity with XML specifications, some element tags can include attributes. Unless otherwise indicated, these attributes are optional and a default value is used when they are omitted. The syntax of elements with attributes is as follows:

- Double tags: `<tag attribute="value"> </macro>`
- Single tags: `<tag attribute="value"/>`

If the element accepts several attributes, you can group them in the same line of command, separated by a space: `\<tag attribute1="value" attribute2="value" attribute3="value" ... >`

Here is the list of tags and their mode of use:

| Element tags  | Description   |
|---|---|
| <code>&lt;macros&gt;</code><br><code>&lt;/macros&gt;</code> | Start and end of macro file (mandatory tag).  |
| <code>&lt;macro&gt;</code><br><code>&lt;/macro&gt;</code>   | Start and end of the definition of a macro and its attributes.  |
|   | <i>Attributes:</i>  |
|   | - name: Name** of macro as it appears in menus and Code Editor lists (mandatory attribute).   |
|   | - type_ahead_text: Character string* to be entered to call the macro using the type-ahead (aka autocomplete) function.              |
|   | - in_menu: Boolean indicating whether the macro can be called using the context menu*. Values = "true" (default) or "false."        |
|   | - type_ahead: Boolean indicating whether the macro can be called using the type ahead (aka autocomplete) function*. Values = "true" |

## Element tags

using the type-ahead (aka autocomplete) function". Values = "true" (default) or "false."

### Description

- method\_event: Used to trigger the automatic calling of the macro depending on the current handling phase of each method (creation, closing, and so on). Values = "on\_load": The macro is triggered on the opening of each method, "on\_save": The macro is triggered when each method is saved (closing of a modified method or saving using the File>Save command, "on\_create": The macro is triggered when each method is created, "on\_close": The macro is triggered when each method is closed.

"on\_save" and "on\_close" can be used jointly --- in other words, both of these events are generated when a modified method is closed. On the other hand, "on\_create" and "on\_load" are never generated in a consecutive manner. This attribute can be used, for example, to preformat methods when they are created (comments in header area) or to record information such as the date and time when they are closed.

- version: Used to activate the new mode of supporting text selections for the macro (see the "About the <method> Tag" section below). To activate this new mode, pass the value "2". If you omit this attribute or pass version="1", the former mode is kept.

- in\_toolbar: Boolean indicating if the macro must be present in the menu of the Macro button of the toolbar. Values= "true" (default) or "false".

<selection/>

Tag replaced by the selected text when the macro is inserted. The selection may be empty.

<text> </text>

Start and end of code that must be inserted in the method. A carriage return will be added before and after the code.

<method>  
</method>

Start and end of the name of the project method and its (optional) parameter. The method is executed when the macro is called. You can pass a parameter in the form ("param1;param2;..."). This parameter will be received in the method using the variables \$1, \$2, etc. For additional information about this tag, refer to the "About the <method> Tag" section below.

<caret/>

Location of the insertion point in the code after the macro has been inserted.

<user\_4D/>

Tag replaced by the name of the current 4D user.

<user\_os/>

Tag replaced by the current system user name.

<method\_name/>

Tag replaced by the current method name.

<method\_path/>

Tag replaced by path syntax (as returned by [METHOD\\_Get\\_path](#) of the current method).

<date/>

Tag replaced by the current date.

#### Attribute:

- format: 4D format used to display the date. If no format is set, the default format is used. Values = number of 4D format (0 to 8).

<time/>

Tag replaced by the current time.

#### Attribute:

- format: 4D format used to display the time. If no format is set, the default format is used. Values = number of 4D format (0 to 6).

<clipboard/>

Tag replaced by the contents of the clipboard.

#### Attribute:

- index: Clipboard to be pasted. Values = number of the clipboard (0 to 9).

- Macros can be called using the context menu of the Code Editor or using the type-ahead function (see the following section).
- If you want to conform to XML language specifications, you must not use extended characters (accented characters, quotation marks, etc.).

Here is an example of a macro definition:

| Content of macro                                 | Comments  |
|--|---|
| <?xml<br>version="1.0" . . . ?>                  | XML declaration   |
| <!DOCTYPE macros<br>SYSTEM>                      | Document declaration  |
| <macros>   | Start of macros XML file  |
| <macro<br>name="RecordLoop">                     | Start of macro definition and name  |
| <text>   | Start of macro code   |
| For(\$i;1;Records in<br>selection(<Selection/>)) | The <Selection/> tag will be replaced by the selected code in the 4D method at the time of macro insertion (for instance, a table name) |
| SAVE   |   |
| RECORD(<Selection/>)                             |   |
| NEXT   |   |
| RECORD(<Selection/>)                             |   |
| End for  |   |
| </text>  | End of macro code   |
| </macro>   | End of macro definition   |
| </macros>  | End of macros XML file  |

## About the `<method>` tag

The `<method>` tag allows you to generate and use macro-commands that execute 4D project methods. This allows developers to create sophisticated functions that can be distributed via macro-commands which are associated with components. For example, the following macro will cause the *MyMethod* method to be executed with the name of the current method as parameter:

```
<method>MyMethod ("<method_name/>")</method>
```

The code of a called method is executed in a new process. This process is killed once the method is executed.

The structure process remains frozen until the called method execution is completed. You must make sure that the execution is quick and that there is no risk of it blocking the application. If this occurs, use the **Ctrl+F8** (Windows) or **Command+F8** (Mac OS) command to "kill" the process.

## Calling macros

By default, macros can be called using the context menu or toolbar of the Code Editor, the autocomplete function, or a specific list at the bottom of the Code Editor window.

Note that for each macro it is possible to restrict the possibility of calling it using the context menu and/or the autocomplete function.

### Context menu and toolbar

By default, all macros can be called via the context menu of the Code Editor (using the **Insert macro** hierarchical command) or the **Macros** button of the toolbar.

The *in\_menu* attribute of the `<macro>` tag is used to set whether or not the macro appears in this menu.

In the context menu, macros are displayed in the order of the "Macros.xml" file and any additional XML files. It is thus possible to change the order by modifying these files.

## Autocomplete

By default, all macros are accessible using the autocomplete (aka type-ahead) function (see [Writing a method](#)). The *type\_ahead* attribute of the `<macro>` tag can be used to exclude a macro from this type of operation.

**Note:** If the macro contains the `<selection/>` tag, it will not appear in the autocomplete pop-up window.

## Code Editor list

You can display your macros in a list of the Code Editor (see [Writing a method](#)). Simply double-click on the name of a macro in the list in order to call it. It is not possible to exclude a specific macro from this list.

## Compatibility notes

Macro support can change from one version of 4D to another. In order to keep the different versions compatible while maintaining your customizations, 4D does not remove any previous versions. If you want to use the latest features available, you must adapt your version accordingly.

## Text selection variables for methods

It is recommended to manage text selections using the [GET MACRO PARAMETER](#) and [SET MACRO PARAMETER](#) commands. These commands can be used to overcome the partitioning of the host project/component execution spaces and thus allow the creation of components dedicated to the management of macros. In order to activate this mode for a macro, you must declare the Version attribute with the value 2 in the Macro element. In this case, 4D no longer manages the predefined variables `_textSel`, `_textReplace`, etc. and the [GET MACRO PARAMETER](#) and [SET MACRO PARAMETER](#) commands are used. This attribute must be declared as follows:

```
<macro name="MyMacro" version="2">
--- Text of the macro ---
</macro>
```

If you do not pass this attribute, the previous mode is kept.

## Incompatibilities related to the XML standard

Strict syntax rules must be observed in order for macros files to respect the XML standard. This may lead to incompatibilities with the code of macros created with previous versions and prevent the loading of XML files. The following are the main sources of malfunctioning:

- Comments of the "/\* my comment" type, allowed inside `<macro>` elements in previous versions of 4D, are not compatible with the XML syntax. The lines of comments must respect the standard "`<!-- my comment -->`" form.

- The `<>` symbols used more particularly for interprocess object names must be encoded. For example, the `<>params` variable must be written `&lt;gt;params`.
- The initial `<macros>` declaration tag could be omitted in previous versions of 4D. It is now mandatory; otherwise, the file will not be loaded.

## Transformation tags

4D provides a set of transformation tags which allow you to insert references to 4D variables or expressions, or to perform different types of processing within a source text, referred to as a "template". These tags are interpreted when the source text is executed and generate an output text.

This principle is used in particular by the 4D Web server to build [web template pages](#).

These tags are generally to be inserted as HTML type comments (`<!--#Tag  
Contents-->`) but an [xml-compliant alternative syntax](#) is available for some of them.

It is possible to mix several types of tags. For example, the following HTML structure is entirely feasible:

```
<HTML>
<BODY>
<!--#4DSCRIPT/PRE_PROCESS-->      (Method call)
<!--#4DIF (myvar=1)-->      (If condition)
    <!--#4DINCLUDE banner1.html-->      (Subpage insertion)
<!--#4DENDIF-->      (End if)
<!--#4DIF (mtvar=2)-->
    <!--#4DINCLUDE banner2.html-->
<!--#4DENDIF-->

<!--#4DLOOP [TABLE]-->      (Loop on the current selection)
<!--#4DIF ([TABLE]ValNum>10)-->      (If [TABLE]ValNum>10)
    <!--#4DINCLUDE subpage.html-->      (Subpage insertion)
<!--#4DELSE-->      (Else)
    <B>Value: <!--#4DTEXT [TABLE]ValNum--></B><br />      (Field display)
<!--#4DENDIF-->
<!--#4DENDLOOP-->      ] (End for)
</BODY>
</HTML>
```

## Principles for using tags

### Parsing

Parsing the contents of a *template* source is done in two contexts:

- Using the `PROCESS 4D TAGS` command; this command accepts a *template* as input, as well as optional parameters and returns a text resulting from the processing.
- Using 4D's integrated HTTP server: [template pages](#) sent by means of the `WEB SEND FILE` (.htm, .html, .shtm, .shtml), `WEB SEND BLOB` (text/html type BLOB), `WEB SEND TEXT` commands, or called using URLs. In this last case, for reasons of optimization, pages that are suffixed with ".htm" and ".html" are NOT parsed. In order to parse HTML pages in this case, you must add the suffix ".shtm" or ".shtml" (for example, <http://www.server.com/dir/page.shtm>).

### Recursive processing

4D tags are interpreted recursively: 4D always attempts to reinterpret the result of a transformation and, if a new transformation has taken place, an additional interpretation is performed, and so on until the product obtained no longer requires any further transformation. For example, given the following statement:

```
<!--#4DHTML [Mail]Letter_type-->
```

If the `[Mail]Letter_type` text field itself contains a tag, for example `<!--#4DSCRIPT/m_Gender-->`, this tag will be evaluated recursively after the interpretation of the 4DHTML tag.

This powerful principle meets most needs related to text transformation. Note, however, that in some cases this can also allow malicious code to be inserted in the web context, [which can be avoided](#).

## Identifiers with tokens

To ensure the correct evaluation of expressions processed via tags, regardless of the language or 4D version, it's recommended to use the tokenized syntax for elements whose name may vary over versions (commands, tables, fields, constants). For example, to insert the `Current time` command, enter `Current time:C178`.

## Using the ":" as decimal separator

4D always uses the period character (.) as a decimal separator when evaluating a numerical expression using a 4D tag `4DTEXT`, `4DHTML`, and `4DEVAL`. Regional settings are ignored. This feature facilitates code maintenance and compatibility between 4D languages and versions.

# 4DBASE

**Syntax:** `<!--#4DBASE folderPath-->`

The `<!--#4DBASE -->` tag designates the working directory to be used by the `<!--#4DINCLUDE-->` tag.

When it is called in a Web page, the `<!--#4DBASE -->` tag modifies all subsequent `<!--#4DINCLUDE-->` calls on this page, until the next `<!--.....-->`, if any. If the `<!--#4DBASE -->` folder is modified from within an included file, it retrieves its original value from the parent file.

The `folderPath` parameter must contain a pathname relative to the current page and it must end with a slash (/). The designated folder must be located inside the Web folder.

Pass the "WEBFOLDER" keyword to restore the default path (relative to the page).

The following code, which must specify a relative path for each call:

```
<!--#4DINCLUDE subpage.html-->
<!--#4DINCLUDE folder/subpage1.html-->
<!--#4DINCLUDE folder/subpage2.html-->
<!--#4DINCLUDE folder/subpage3.html-->
<!--#4DINCLUDE ../folder/subpage.html-->
```

... is equivalent to:

```
<!--#4DINCLUDE subpage.html-->
<!--#4DBASE folder/-->
<!--#4DINCLUDE subpage1.html-->
<!--#4DINCLUDE subpage2.html-->
<!--#4DINCLUDE subpage3.html-->
<!--#4DBASE ../folder/-->
<!--#4DINCLUDE subpage.html-->
<!--#4DBASE WEBFOLDER-->
```

For example, to set a directory for the home page:

```
/* Index.html */
<!--#4DIF LangFR=True-->
    <!--#4DBASE FR/-->
<!--#4ELSE-->
    <!--#4DBASE US/-->
<!--#4DENDIF-->
<!--#4DINCLUDE head.html-->
<!--#4DINCLUDE body.html-->
<!--#4DINCLUDE footer.html-->
```

In the "head.html" file, the current folder is modified through `<!--#4DBASE -->`, without this changing its value in "Index.html":

```
/* Head.htm */
/* the working directory here is relative to the included file (FR/ or
US/) */
<!--#4DBASE Styles/-->
<!--#4DINCLUDE main.css-->
<!--#4DINCLUDE product.css-->
<!--#4DBASE Scripts/-->
<!--#4DINCLUDE main.js-->
<!--#4DINCLUDE product.js-->
```

## 4DCODE

**Syntax:** `<!--#4DCODE codeLines-->`

The `4DCODE` tag allows you to insert a multi-line 4D code block in a template.

When a `<!--#4DCODE` sequence is detected that is followed by a space, a CR or a LF character, 4D interprets all the lines of code up to the next `-->` sequence. The code block itself can contain carriage returns, line feeds, or both; it will be interpreted sequentially by 4D.

For example, you can write in a template:

```
<!--#4DCODE
//PARAMETERS initialization
C_OBJECT:C1216($graphParameters)
OB SET:C1220($graphParameters;"graphType";1)
$graphType:=1
//...your code here
If(OB Is defined:C1231($graphParameters;"graphType"))
    $graphType:=OB GET:C1224($graphParameters;"graphType")
    If($graphType=7)
        $nbSeries:=1
        If($nbValues>8)
            DELETE FROM ARRAY:C228 ($yValuesArrPtr{1}->;9;100000)
            $nbValues:=8
        End if
    End if
End if
-->
```

Here are the 4DCODE tag features:

- The `TRACE` command is supported and activates the 4D debugger, thus allowing you to debug your template code.
- Any error will display the standard error dialog that lets the user stop code execution or enter debugging mode.

- The text in between `<!--#4DCODE` and `-->` is split into lines accepting any line-ending convention (cr, lf, or crlf).
- The text is tokenized within the context of the database that called `PROCESS 4D TAGS`. This is important for recognition of project methods for example. The [Available through tags and 4D URLs \(4DACTION ...\)](#) method property is not taken into account.
- Even if the text always uses English-US, it is recommended to use the token syntax (:Cxxx) for command and constant names to protect against potential problems due to commands or constants being renamed from one version of 4D to another.

The fact that 4DCODE tags can call any of the 4D language commands or project methods could be seen as a security issue, especially when the database is available through HTTP. However, since it executes server-side code called from your own template files, the tag itself does not represent a security issue. In this context, as for any Web server, security is mainly handled at the level of remote accesses to server files.

## 4DEACH and 4DENDEACH

**Syntax:** `<!--#4DEACH variable in expression--> <!--#4DENDEACH-->`

The `<!--#4DEACH-->` comment allows iterating a specified item over all values of the *expression*. The item is set to a *variable* whose type depends on the *expression* type.

The `<!--#4DEACH-->` comment can iterate through three expression types:

- [collections](#): loop through each element of the collection,
- [entity selections](#): loop through each entity,
- [objects](#): loop through each object property.

The number of iterations is evaluated at startup and will not change during the processing. Adding or removing items during the loop is usually not recommended since it may result in missing or redundant iterations.

`<!--#4DEACH item in collection-->`

This syntax iterates on each *item* of the *collection*. The code portion located between `<!--#4DEACH -->` and `<!--#4DENDEACH-->` is repeated for each collection element.

The *item* parameter is a variable of the same type as the collection elements.

The collection must contain only **elements of the same type**, otherwise an error is returned as soon as the *item* variable is assigned the first mismatched value type.

The number of loops is based on the number of elements of the collection. At each iteration, the *item* variable is automatically filled with the matching element of the collection. The following points must be taken into account:

- If the *item* variable is of the object type or collection type (i.e. if *expression* is a collection of objects or of collections), modifying this variable will automatically modify the matching element of the collection (because objects and collections share the same references). If the variable is of a scalar type, only the variable will be modified.
- The *item* variable gets the same type as the first collection element. If any collection element is not of the same type as the variable, an error is generated and the loop stops.
- If the collection contains elements with a Null value, an error is generated if the

*item* variable type does not support Null values (such as longint variables).

### Example with a collection of scalar values

*getNames* returns a collection of strings. The method has been declared as "[available through 4D tags and URLs](#)".

```
<table class="table">

<tr><th>Name</th></tr>

    <!--#4DEACH $name in getNames-->
<tr>
    <td><!--#4DTEXT $name--></td>
</tr>
    <!--#4DENDEACH-->
</table>
```

### Example with a collection of objects

*getSalesPersons* returns a collection of objects.

```
<table class="table">
    <!--#4DCODE
        $salePersons:=getSalesPersons
    -->
<tr><th>ID</th><th>Firstname</th><th>Lastname</th></tr>

    <!--#4DEACH $salesPerson in $salePersons-->
<tr>
    <td><!--#4DTEXT $salesPerson.ID--></td>
    <td><!--#4DTEXT $salesPerson.firstname--></td>
    <td><!--#4DTEXT $salesPerson.lastname--></td>
</tr>
    <!--#4DENDEACH-->
</table>

<!--#4DEACH entity in entitySelection-->
```

This syntax iterates on each *entity* of the *entitySelection*. The code portion located between `<!--#4DEACH -->` and `<!--#4DENDEACH-->` is repeated for each entity of the entity selection.

The *entity* parameter is an object variable of the entity selection class.

The number of loops is based on the number of entities of the entity selection. At each iteration, the *entity* object variable is automatically filled with the matching entity of the entity selection.

### Example with a html table

```

<table class="table">

    <tr><th>ID</th><th>Name</th><th>Total purchase</th></tr>

        <!--#4DEACH $customer in ds.Customers.all()-->
        <tr>
            <td><!--#4DTEXT $customer.ID--></td>
            <td><!--#4DTEXT $customer.name--></td>
            <td><center><!--#4DTEXT
String($customer.totalPurchase;"$###,##0")--></center></td>
        </tr>
        <!--#4DENDEACH-->
    </table>

```

### Example with PROCESS 4D TAGS

```

var customers : cs.CustomersSelection
var $input; $output : Text

customers:=ds.Customers.all()
$input:="<!--#4DEACH $cust in customers-->"
$input:=$input+"<!--#4DTEXT $cust.name -->"+Char(Carriage return)
$input:=$input+"<!--#4DENDEACH-->"
PROCESS 4D TAGS($input; $output)
TEXT TO DOCUMENT("customers.txt"; $output)

```

<!--#4DEACH property in object-->

This syntax iterates on each *property* of the *object*. The code portion located between `<!--#4DEACH -->` and `<!--#4DENDEACH-->` is repeated for each property of the object.

The *property* parameter is a text variable automatically filled with the name of the currently processed property.

The properties of the object are processed according to their creation order. During the loop, properties can be added to or removed from the object, without modifying the number of loops that will remain based on the original number of properties of the object.

### Example with the properties of an object

`getGamers` is a project method that returns an object like ("Mary"; 10; "Ann"; 20; "John"; 40) to figure gamer scores.

```

<table class="table">
    <!--#4DCODE
    $gamers:=getGamers
    -->

    <tr><th>Gamers</th><th>Scores</th></tr>

        <!--#4DEACH $key in $gamers-->
        <tr>
            <td><!--#4DTEXT $key--></td>
            <td><!--#4DTEXT $gamers[$key]--></td>
        </tr>
        <!--#4DENDEACH-->
    </table>

```

## 4DEVAL

**Syntax:** <!--#4DEVAL expression-->

**Alternative syntax:** \$4DEVAL(expression)

The `4DEVAL` tag allows you to assess a 4D variable or expression. Like the `4DHTML` tag, `4DEVAL` does not escape HTML characters when returning text. However, unlike `4DHTML` or `4DTEXT`, `4DEVAL` allows you to execute any valid 4D statement, including assignments and expressions that do not return any value.

For example, you can execute:

```
$input:="<!--#4DEVAL a:=42-->" //assignment  
$input:=$input+"<!--#4DEVAL a+1-->" //calculation  
PROCESS 4D TAGS($input;$output)  
//$output = "43"
```

In case of an error during interpretation, the text inserted will be in the form: <!--#4DEVAL expr-->: ## error # error code.

For security reasons, it is recommended to use the `4DTEXT` tag when processing data introduced from outside the application, in order to prevent the [insertion of malicious code](#).

## 4DHTML

**Syntax:** <!--#4DHTML expression-->

**Alternative syntax:** \$4DHTML(expression)

Just like the `4DTEXT` tag, this tag lets you assess a 4D variable or expression that returns a value, and insert it as an HTML expression. Unlike the `4DTEXT` tag, this tag does not escape HTML special characters (e.g. ">").

For example, here are the processing results of the 4D text variable myvar with the available tags:

myvar Value	Tags	Result
myvar:="<B>"<!--#4DTEXT myvar-->&lt;B&gt;;	&lt;B&gt;;	&lt;B&gt;;
myvar:="<B>"<!--#4DHTML myvar--><B>	<B>	<B>

In case of an interpretation error, the inserted text will be <!--#4DHTML myvar--> : ## error # error code.

For security reasons, it is recommended to use the `4DTEXT` tag when processing data introduced from outside the application, in order to prevent the [insertion of malicious code](#).

## 4DIF, 4ELSE, 4ELSEIF and 4DENDIF

**Syntax:** <!--#4DIF expression--> {<!--#4ELSEIF expression2-->...<!--#4ELSEIF expressionN-->} {<!--#4ELSE-->} <!--#4DENDIF-->

Used with the `<!--#4ELSEIF-->` (optional), `<!--#4ELSE-->` (optional) and `<!--#4DENDIF-->` comments, the `<!--#4DIF expression-->` comment offers the possibility to execute portions of code conditionally.

The `expression` parameter can contain any valid 4D expression returning a Boolean

value. It must be indicated within parenthesis and comply with the 4D syntax rules.

The `<!--#4DIF expression--> ... <!--#4DENDIF-->` blocks can be nested in several levels. Like in 4D, each `<!--#4DIF expression-->` must match a `<!--#4DENDIF-->`.

In case of an interpretation error, the text "`<!--#4DIF expression-->`: A Boolean expression was expected" is inserted instead of the contents located between `<!--#4DIF -->` and `<!--#4DENDIF-->`. Likewise, if there are not as many `<!--#4DENDIF-->` as `<!--#4DIF -->`, the text "`<!--#4DIF expression-->`: 4DENDIF expected" is inserted instead of the contents located between `<!--#4DIF -->` and `<!--#4DENDIF-->`.

Using the `<!--#4DELSEIF-->` tag, you can test an unlimited number of conditions. Only the code that follows the first condition evaluated as `True` is executed. If no conditions are true, no statement is executed (if there is no final `<!--#4DELSE-->`). You can use a `<!--#4DELSE-->` tag after the last `<!--#4DELSEIF-->`. If all the conditions are false, the statements following the `<!--#4DELSE-->` are executed.

The two following codes are equivalent.

Code using `4DELSE` only:

```
<!--#4DIF Condition1-->
/* Condition1 is true*/
<!--#4DELSE-->
    <!--#4DIF Condition2-->
        /* Condition2 is true*/
    <!--#4DELSE-->
        <!--#4DIF Condition3-->
            /* Condition3 is true */
        <!--#4DELSE-->
            /*None of the conditions are true*/
        <!--#4DENDIF-->
    <!--#4DENDIF-->
<!--#4DENDIF-->
```

Similar code using the `4DELSEIF` tag:

```
<!--#4DIF Condition1-->
/* Condition1 is true*/
<!--#4DELSEIF Condition2-->
/* Condition2 is true*/
<!--#4DELSEIF Condition3-->
/* Condition3 is true */
<!--#4DELSE-->
/* None of the conditions are true*/
<!--#4DENDIF-->
```

This example of code inserted in a static HTML page displays a different label according the `vname#""` expression result:

```
<BODY>
...
<!--#4DIF (vname# "")-->
Names starting with <!--#4DTEXT vname-->.
<!--#4DELSE-->
No name has been found.
<!--#4DENDIF-->
...
</BODY>
```

This example inserts different pages depending on which user is connected:

```
<!--#4DIF LoggedIn=False-->
    <!--#4DINCLUDE Login.htm -->
<!--#4ELSEIF User="Admin" -->
    <!--#4DINCLUDE AdminPanel.htm -->
<!--#4ELSEIF User="Manager" -->
    <!--#4DINCLUDE SalesDashboard.htm -->
<!--#4ELSE-->
    <!--#4DINCLUDE ItemList.htm -->
<!--#4DENDIF-->
```

## 4DINCLUDE

**Syntax:** `<!--#4DINCLUDE path-->`

This tag is mainly designed to include an HTML page (indicated by the *path* parameter) in another HTML page. By default, only the body of the specified HTML page, i.e. the contents found within the `<body>` and `</body>` tags, is included (the tags themselves are not included). This lets you avoid conflicts related to meta tags present in the headers.

However, if the HTML page specified does not contain `<body>` `</body>` tags, the entire page is included. It is up to you to verify the consistency of the meta tags.

The `<!--#4DINCLUDE -->` comment is very useful for tests (`<!--#4DIF-->`) or loops (`<!--#4DLOOP-->`). It is very convenient to include banners according to a criteria or randomly. When including, regardless of the file name extension, 4D analyzes the called page and then inserts the contents (modified or not) in the page originating the `4DINCLUDE` call.

An included page with the `<!--#4DINCLUDE -->` comment is loaded in the Web server cache the same way as pages called via a URL or sent with the `WEB SEND FILE` command.

In *path*, put the path leading to the document to include. Warning: In the case of a `4DINCLUDE` call, the path is relative to the document being analyzed, that is, the "parent" document. Use the slash character (/) as a folder separator and the two dots (..) to go up one level (HTML syntax). When you use the `4DINCLUDE` tag with the `PROCESS 4D TAGS` command, the default folder is the project folder.

You can modify the default folder used by the `4DINCLUDE` tag in the current page, using the `<!--#4DBASE -->` tag (see below).

The number of `<!--#4DINCLUDE path-->` within a page is unlimited. However, the `<!--#4DINCLUDE path-->` calls can be made only at one level. This means that, for example, you cannot insert `<!--#4DINCLUDE mydoc3.html-->` in the *mydoc2.html* body page, which is called by `<!--#4DINCLUDE mydoc2-->` inserted in *mydoc1.html*. Furthermore, 4D verifies that inclusions are not recursive.

In case of error, the inserted text is "`<!--#4DINCLUDE path-->` :The document cannot be opened".

Examples:

```
<!--#4DINCLUDE subpage.html-->
<!--#4DINCLUDE folder/subpage.html-->
<!--#4DINCLUDE ../folder/subpage.html-->
```

## 4DLOOP and 4DENDLOOP

Syntax: <!--#4DLOOP condition--> <!--#4DENDLOOP-->

This comment allows repetition of a portion of code as long as the condition is fulfilled. The portion is delimited by <!--#4DLOOP--> and <!--#4DENDLOOP-->.

The <!--#4DLOOP condition--> ... <!--#4DENDLOOP--> blocks can be nested. Like in 4D, each <!--#4DLOOP condition--> must match a <!--#4DENDLOOP-->.

There are five kinds of conditions:

<!--#4DLOOP [table]-->

This syntax makes a loop for each record from the table current selection in the current process. The code portion located between the two comments is repeated for each current selection record.

When the `4DLOOP` tag is used with a table, records are loaded in "Read only" mode.

The following code:

```
<!--#4DLOOP [People]-->
<!--#4DTEXT [People]Name--> <!--#4DTEXT [People]Surname--><br/>
<!--#4DENDLOOP-->
```

... could be expressed in 4D language in the following way:

```
FIRST RECORD([People])
While(Not(End selec tion([People])))
  ...
  NEXT RECORD([People])
End while
```

<!--#4DLOOP array-->

This syntax makes a loop for each array item. The array current item is increased when each code portion is repeated.

This syntax cannot be used with two dimension arrays. In this case, it is better to combine a method with nested loops.

The following code example:

```
<!--#4DLOOP arr_names-->
<!--#4DTEXT arr_names{arr_names}--><br/>
<!--#4DENDLOOP-->
```

... could be expressed in 4D language in the following way:

```
For($Elem;1;Size of array(arr_names))
  arr_names:=$Elem
  ...
End for
```

<!--#4DLOOP method-->

This syntax makes a loop as long as the method returns `True`. The method takes a Long Integer parameter type. First it is called with the value 0 to allow an initialization stage (if necessary); it is then called with the values 1 ,then 2, then 3 and so on, as

long as it returns `True`.

For security reasons, within a Web process, the `On Web Authentication` database method can be called once just before the initialization stage (method execution with 0 as parameter). If the authentication is OK, the initialization stage will proceed.

`C_BOOLEAN($0)` and `C_LONGINT($1)` MUST be declared within the method for compilation purposes.

The following code example:

```
<!--#4DLOOP my_method-->
<!--#4DTEXT var--> <br/>
<!--#4DENDLOOP-->
```

... could be expressed in 4D language in the following way:

```
If (AuthenticationWebOK)
  If (my_method(0))
    $counter:=1
    While (my_method($counter))
      ...
      $counter:=$counter+1
    End while
  End if
End if
```

The `my_method` method can be as follows:

```
C_LONGINT($1)
C_BOOLEAN($0)
If ($1=0) `Initialisation
  $0:=True
Else
  If ($1<50)
    ...
    var:...
    $0:=True
  Else
    $0:=False `Stops the loop
  End if
End if
```

```
<!--#4DLOOP expression-->
```

With this syntax, the `4DLOOP` tag makes a loop as long as the *expression* returns `True`. The expression can be any valid Boolean expression and must contain a variable part to be evaluated in each loop to avoid infinite loops.

For example, the following code:

```
<!--#4DEVAL $i:=0-->
<!--#4DLOOP ($i<4)-->
<!--#4DEVAL $i-->
<!--#4DEVAL $i:=$i+1-->
<!--#4DENDLOOP-->
```

...produces the following result:

```
0  
1  
2  
3
```

```
<!--#4DLOOP pointerArray-->
```

In this case, the `4DLOOP` tag works like it does with an array: it makes a loop for each element of the array referenced by the pointer. The current array element is increased each time the portion of code is repeated.

This syntax is useful when you pass an array pointer as a parameter to the `PROCESS 4D TAGS` command.

Example:

```
ARRAY TEXT($array;2)  
$array{1}:="hello"  
$array{2}:="world"  
$input:="<!--#4DEVAL $1-->"  
$input:=$input+"<!--#4DLOOP $2-->"  
$input:=$input+"<!--#4DEVAL $2->{$2->}--> "  
$input:=$input+"<!--#4DENDLOOP-->"  
PROCESS 4D TAGS($input;$output;"elements = ";->$array)  
// $output = "elements = hello world "
```

In case of an interpretation error, the text "`<!--#4DLOOP expression-->: description`" is inserted instead of the contents located between `<!--#4DLOOP -->` and `<!--#4DENDLOOP-->`.

The following messages can be displayed:

- Unexpected expression type (standard error);
- Incorrect table name (error on the table name);
- An array was expected (the variable is not an array or is a two dimension array);
- The method does not exist;
- Syntax error (when the method is executing);
- Access error (you do not have the appropriate access privileges to access the table or the method).
- `4DENDLOOP` expected (the `<!--#4DENDLOOP-->` number does not match the `<!--#4DLOOP -->`).

## 4SCRIPT/

Syntax: `<!--#4SCRIPT/MethodName/MyParam-->`

The `4SCRIPT` tag allows you to execute 4D methods when processing the template. The presence of the `<!--#4SCRIPT/MethodName/MyParam-->` tag as an HTML comment launches the execution of the `MethodName` method with the `Param` parameter as a string in `$1`.

If the tag is called in the context of a Web process, when the page is loaded, 4D calls the `On Web Authentication` database method (if it exists). If it returns True, 4D executes the method.

The method must return text in `$0`. If the string starts with the code character 1, it is considered as HTML (the same principle is true for the `4DHTML` tag).

For example, let's say that you insert the following comment "`<!--#4DSCRIPT/MYMETH/MYPARAM-->`" into a template Web page. When loading the page, 4D calls the `On Web Authentication` database method, then calls the `MYMETH` method and passes the string `"/MYPARAM"` as the parameter `$1`. The method returns text in `$0` (for example `"12/31/21"`); the expression "`Today is <!--#4DSCRIPT/MYMETH/MYPARAM-->`" therefore becomes `"Today is 12/31/21"`.

The `MYMETH` method is as follows:

```
//MYMETH  
C_TEXT($0;$1) //These parameters must always be declared  
$0:=String(Current date)
```

A method called by `4DSCRIPT` must not call interface elements (`DIALOG`, `ALERT`, etc.).

As 4D executes methods in their order of appearance, it is absolutely possible to call a method that sets the value of many variables that are referenced further in the document, whichever mode you are using. You can insert as many `<!--#4DSCRIPT...-->` comments as you want in a template.

## 4DTEXT

**Syntax:** `<!--#4DTEXT expression-->`

**Alternative syntax:** `$4DTEXT(expression)`

The tag `<!--#4DTEXT expression-->` allows you to insert a reference to a 4D variable or expression returning a value. For example, if you write (in an HTML page):

```
<P>Welcome to <!--#4DTEXT vtSiteName-->!</P>
```

The value of the 4D variable `vtSiteName` will be inserted in the HTML page when it is sent. This value is inserted as simple text, special HTML characters such as `">"` are automatically escaped.

You can also insert 4D expressions. You can for example directly insert the contents of a field (`<!--#4DTEXT [tableName]fieldName-->`), an array element (`<!--#4DTEXT tabarr{1}-->`) or a method returning a value (`<!--#4DTEXT mymethod-->`). The expression conversion follows the same rules as the variable ones. Moreover, the expression must comply with 4D syntax rules.

For security reasons, it is recommended to use this tag when processing data introduced from outside the application, in order to prevent the [insertion of malicious code](#).

In case of an evaluation error, the inserted text will appear as `<!--#4DTEXT myvar-->`: `## error # error code.`

- You must use process variables.
- You can display the content of a picture field. However, it is not possible to display the content of a picture array item.
- It is possible to display the contents of an object field by means of a 4D formula. For example, you can write `<!--#4DTEXT OB Get:C1224([Rect]Desc;\\"color\\")-->`.
- You will usually work with Text variables. However, you can also use BLOB variables. You just need to generate BLOBS in `Text without length` mode.

## Alternative syntax for 4DTEXT, 4DHTML, 4DEVAL

Several existing 4D transformation tags can be expressed using a \$-based syntax:

### \$4dtag (expression)

can be used instead of

```
<!--#4dtag expression-->
```

This alternative syntax is available only for tags used to return processed values:

- [4DTEXT](#)
- [4DHTML](#)
- [4DEVAL](#)

(Other tags, such as 4DIF or 4DSCRIPT, must be written with the regular syntax).

For example, you can write:

```
$4DEVAL (UserName)
```

instead of:

```
<!--#4DEVAL (UserName) -->
```

The main advantage of this syntax is that it allows you to write XML-compliant templates. Some 4D developers need to create and validate XML-based templates using standard XML parser tools. Since the "<" character is invalid in an XML attribute value, it was not possible to use the "<!-- -->" syntax of 4D tags without breaking the document syntax. On the other hand, escaping the "<" character will prevent 4D from interpreting the tags correctly.

For example, the following code would cause an XML parsing error because of the first "<" character in the attribute value:

```
<line x1="" y1=""/>
```

Using the \$ syntax, the following code is validated by the parser:

```
<line x1="$4DEVAL($x)" y1="$4DEVAL($graphY1)"/>
```

Note that `$4dtag` and `<--#4dtag -->` are not strictly equivalent: unlike `<--#4dtag -->`, `$4dtag` processing does not interpret 4D tags [recursively](#). `$` tags are always evaluated once and the result is considered as plain text.

The reason for this difference is to prevent malicious code injection. As [explained below](#), it is strongly recommended to use `4DTEXT` tags instead of `4DHTML` tags when handling user text to protect against unwanted reinterpretation of tags: with `4DTEXT`, special characters such as "<" are escaped, thus any 4D tags using the `<!--#4dtag expression -->` syntax will lose their particular meaning. However, since `4DTEXT` does not escape the `$` symbol, we decided to break support for recursion in order to prevent malicious injection using the `$4dtag (expression)` syntax.

The following examples show the result of processing depending on the syntax and tag used:

```

// example 1
myName:="<!--#4DHTML QUIT 4D-->" //malicious injection
input:="My name is: <!--#4DHTML myName-->"
PROCESS 4D TAGS(input;output)
//4D will quit!
// example 2
myName:="<!--#4DHTML QUIT 4D-->" //malicious injection
input:="My name is: <!--#4DTEXT myName-->"
PROCESS 4D TAGS(input;output)
//output is "My name is: <!--#4DHTML QUIT 4D-->"
// example 3
myName:="$4DEVAL(QUIT 4D)" //malicious injection
input:="My name is: <!--#4DTEXT myName-->"
PROCESS 4D TAGS(input;output)
//output is "My name is: $4DEVAL(QUIT 4D)"

```

Note that the `$4dtag` syntax supports matching pairs of enclosed quotes or parenthesis. For example, suppose that you need to evaluate the following complex (unrealistic) string:

`String(1) + "\\"(hello)\\\""`

You can write:

```

input:="$4DEVAL( String(1)+\"\\\"(hello)\\\"\\\")"
PROCESS 4D TAGS(input;output)
-->output is 1"(hello)"

```

# Debugging

Errors are common. It would be unusual to write a substantial number of lines of code without generating any errors. Conversely, treating and/or fixing errors is normal, too!

The 4D development environment provides several debugging tools for all types of errors.

## Error types

### Typing errors

Typing errors are detected by the Code Editor. They are displayed in red and additional information is provided at the bottom of the window. Here's a typing error:

Such typing errors usually cause syntax errors (in the above image, the name of the table is unknown). You get the description of the error when you validate the line of code. When this occurs, fix the typing error and type Enter to validate the fix.

### Syntax Errors

Some errors can be caught only when you execute the method. The [Syntax Error Window](#) appears when an error occurs during code execution. For example:

Expand the **Details** area to display the last error and its number.

### Environmental Errors

Occasionally, there may not be enough memory to create a BLOB. Or, when you access a document on disk, the document may not exist or may already be opened by another application. These environmental errors do not directly occur because of your code or the way you wrote it. Most of the time, these errors are easy to treat with an [error catching method](#) installed using the `ON ERR CALL` command.

### Design or Logic Errors

These are generally the most difficult type of error to find. Except for typing errors, all the error types listed above are to a certain extent covered by the expression "Design or logic error". Use the [Debugger](#) to detect them. For example:

- A *syntax error* may occur when you try to use a variable that is not yet initialized.
- An *environmental error* can occur when you try to open a document, because that document's name is received by a subroutine that did not get the right value as a parameter.

Design or logic errors also include such situations as:

- A record is not properly updated because, while calling `SAVE RECORD`, you forgot to first test whether or not the record was locked.
- A method does not do exactly what you expect, because the presence of an optional parameter is not tested.

Sometimes the piece of code that displays the error may be different than the code

that is actually the origin of the problem.

## Runtime Errors

In Application mode, you might obtain errors that you don't see in interpreted mode. Here's an example:

To quickly find the origin of the problem, reopen the interpreted version of the structure file, open the method and go to the corresponding line.

## Syntax Error Window

The Syntax error window automatically appears when the execution of a method is interrupted. This can happen when:

- an error prevents further code execution
- the method produces a false assertion (see the `ASSERT` command)

The upper text area displays a message describing the error. The lower text area shows the line that was executing when the error occurred; the area where the error occurred is highlighted. The expanded Details section contains the "stack" of errors related to the process.

The syntax error window proposes several options:

- **Edit:** Stops all method execution. 4D switches to the Design environment and the method with the error opens in the Code Editor, allowing you to fix it. Use this option when you immediately recognize the mistake and can fix it without further investigation.
- **Trace:** Enters Trace/Debugger mode. The [Debugger](#) window is displayed. If the current line has only executed partially, you may have to click the **Trace** button several times.
- **Continue:** Execution continues. The line with the error may be partially executed, depending on where the error is located. Continue with caution: the error may prevent the rest of your method from executing properly. We recommend clicking **Continue** only if the error is in a trivial call (such as `SET WINDOW TITLE`) that does not prevent executing and testing the rest of your code.

Tip: To ignore an error that occurs repeatedly (for example, in loops), you can turn the **Continue** button into an **Ignore** button. Hold down **Alt** (Windows) or **Option** (macOS) key and click the **Continue** button the first time it appears. The button label changes to **Ignore** if the dialog is called again for the same error.

- **Abort:** Stops method execution and returns to the state before the method started executing:
  - If a form method or object method is executing in response to an event, it is stopped and you return to the form.
  - If the method is executing from within the Application environment, you return to that environment.
- **Copy:** Copies the debugging information into the clipboard. The info describes the

internal environment of the error (number, internal component, etc.). It is formatted as tabbed text.

- **Save...**: Saves the contents of the syntax error window and the call chain in a `.txt` file.

## Debugger

A common beginner mistake in dealing with error detection is to click **Abort** in the Syntax Error Window, go back to the Code Editor, and try to figure out what's going by looking at the code. Do not do that! You will save plenty of time and energy by always using the **Debugger**.

The Debugger allows you to step through methods slowly. It displays all the information you need in order to understand why an error occurred. Once you have this information, you know how to fix the error.

Another reason to use the Debugger is for developing code. Sometimes you may write an algorithm that is more complex than usual. Despite all feelings of accomplishment, you can't be totally sure that your coding is 100% correct. Instead of running it "blind", you can use the `TRACE` command at the beginning of your code, then execute it step by step to keep an eye on what happens.

## Breaks

In the debugging process, you may need to skip the tracing of some parts of the code until a certain line. Or, you may want to trace the code when a given expression has a certain value (e.g. `"$myVar > 1000"`), or every time a specific 4D command is called.

These needs are covered by **breakpoints** and **command catching** features. They can be configured from the Code Editor, the debugger, or the Runtime Explorer.

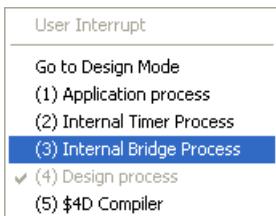
# Debugger

The debugger is useful when you need to spot errors or monitor the execution of methods. It allows you to step through your code slowly and examine the information. This process is called "tracing".

## Calling the Debugger

There are multiple ways to get the Debugger to display:

- Clicking the **Trace** button in the [Syntax Error window](#)
- Using the [TRACE](#) command
- Clicking the **Debug** button in the Execute Method window or selecting **Run and debug...** button in the Code Editor
- Using **Alt+Shift+Right click** (Windows) or **Ctrl+Option+Cmd+Click** (macOS) while a method is executing, then selecting the process to trace in the pop-up menu:



- Clicking the **Trace** button when a process is selected in the Process page of the Runtime Explorer.
- Adding a break point in the Code Editor window or in the Break and Catch pages of the Runtime Explorer.

When called, the debugger window provides the name of the method or class function you're currently tracing, and the action causing the initial appearance of the Debugger window. For example, in the above debugger window:

- *Clients\_BuildLogo* is the method being traced
- The debugger window appeared because it detected a call to the `C_PICTURE` command and this command was one of the commands to be caught

Displaying a new debugger window uses the same configuration as the last window displayed in the same session. If you run several user processes, you can trace them independently and have one debugger window open for each process.

The Debugger window is usually displayed on the machine where the code is executed. With a single-user application, it is always displayed on the machine running the application. With a client/server application, it is displayed:

- on the remote 4D for code running locally
- on the server machine for code running on the server (for example, a method with the **execute on server** option).

If the server is running headless, no debugger window can be displayed on the server, you need to use the remote debugger. See [Debugging from remote machines](#).

## Tool bar Buttons

The debugger's tool bar includes several buttons, associated with default shortcuts:

Default shortcuts can be customized in the Shortcuts Page of the Preferences dialog box.

### No Trace

Tracing stops and normal method execution resumes.

**Shift + F5** or **Shift** + clicking the **No Trace** button resumes execution. It also disables all the subsequent TRACE calls for the current process.

### Step Over

Executes the current method line, indicated by the program counter (the yellow arrow). The Debugger steps to the next line.

The Step Over button does not step into subroutines and functions, it stays at the level of the method you're currently tracing. If you want to also trace subroutines and functions calls, use the **Step Into** button.

In remote debugging, if the method executes on the server, the parent method is called after the last line of the child method executes. If the parent method is executed on the remote side, the **Step Over** button has the same effect as the **No Trace** button.

### Step Into

When a line that calls another method (subroutine or function) is executed, click this button to display the the other method and step through it.

The new method becomes the current (top) method in the [Call Chain Pane](#) of the Debugger window.

When executing a line that does not call another method, this button has the same effect as the **Step Over** button.

### Abort

Stops method execution, and returns to the state before the method started executing:

- When tracing a form or object method executing in response to an event: Stops and returns to the form.
- When tracing a method executing from within the Application environment: Stops and returns to the environment.

### Abort and Edit

Pauses method execution. The method that is executing when you click the **Abort and Edit** button opens in the Code Editor.

**Tip:** Use this button when you know which changes are required in your code, and when these changes are required to pursue the testing of your

methods. After you're finished with the changes, rerun the method.

## Edit

Pauses method execution. The method that is executing at the time you click the Edit button opens in the Code Editor.

If you use this button to modify a method, the modifications are only effective the next time it executes.

**Tip:** Use this button when you know which changes are required in your code and when they don't interfere with the rest of the code to be executed or traced.

## Save Settings

Saves the current configuration of the debugger window and makes it the default configuration. This includes:

- the size and position of the window
- the position of the division lines and the contents of the area that evaluates the expressions

These parameters are stored in the project.

This action is not available in remote debugging mode (see [Debugging from Remote Machines](#)).

## Watch Pane

The **Watch pane** is displayed in the top left corner of the Debugger window, below the Execution Control Tool Bar. Here is an example:

This pane is not available in remote debugging mode.

The **Watch Pane** displays useful general information about the system, the 4D environment, and the execution environment.

The **Expression** column displays the names of the objects and expressions. The **Value** column displays their current corresponding values. Clicking on any value on the right side of the pane allows you to modify the value of the object, if this is permitted for that object.

At any point, you can drag and drop themes, theme sublists (if any), and theme items to the [Custom Watch Pane](#).

## Expression list

### Line Objects

This theme lets you keep track of the values of the objects or expressions:

- used in the line of code to be executed (the one marked with the program counter—the yellow arrow in the [Source Code Pane](#)),
- used in the previous line of code

Since the previous line of code is the one that was just executed before, this theme

therefore shows the objects or expressions of the current line before and after that the line was executed. Let's say you execute the following method:

```
TRACE
$a:=1
$b:=a+1
$c:=a+b
```

1. A Debugger window opens with the program counter set to the line with `a:=1`. At this point the **Line Objects** theme displays:

### \$a Undefined

The `$a` variable is not yet initialized, but it is displayed because it is used in the line to be executed.

2. You click the **Step Over** button. The program counter is now set to the line `b:=a+1`. At this point, the theme displays:

```
$a    1
$b Undefined
```

The value of the `$a` variable is now 1. The `$b` variable is not yet initialized, but it is displayed because it is used in the line to be executed.

3. You click the **Step Over** button again. The program counter is now set on the line with `c:=a+b`. At this point the Line Objects theme displays:

### \$c Undefined

```
$a 1
$b 2
```

The value of the `$b` variable is now 2. The `$c` variable is not yet initialized, but it is displayed because it is used in the line to be executed.

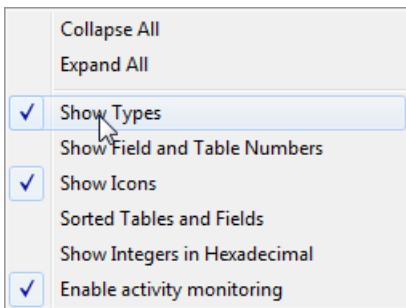
## Variables

This theme is composed of the following subthemes:

Subtheme	Description	Can the values be modified?
Interprocess	List of interprocess variables being used at this point	Yes
Process	List of process variables used by the current process	Yes
Local	List of local variables used by the method being traced	Yes
Parameters	List of parameters received by the method	Yes
Self	Pointer to the current object, when tracing an Object Method	No

Arrays, like other variables, appear in the Interprocess, Process, and Local subthemes, depending on their scope. The debugger displays the first 100 elements. Inside the **Value** column, you can modify the values of array elements, but not the size of the arrays.

To display the variable types and their internal names, right click and check the **Show Types** option in the context menu:



Here's the result:

Variable2 ->\$form.4B9.42 : Text	""
vRecNum ->vRecNum : Text	"2 of 2"

## Current Form Values

This theme contains the name of each dynamic object included in the current form, as well as the value of its associated variable:

▼  Current Form Values	
	bCancel
	bDelete
	Button3
	bValidate
	FirstName
	ID
>	List Box1
>	List Box1
	Form: debugger
	0
	0
	1
	0
	"Tony"
	"2"
	0 elements
	Listbox sub objects

Some objects, such as list box arrays, can be presented as two distinct objects, the variable of the object itself and its data source.

## Constants

Like the Constants page of the Explorer window, this theme displays predefined constants provided by 4D. The expressions from this theme cannot be modified.

## Semaphores

This theme lists the local semaphores currently being set. For each semaphore, the Value column provides the name of the process that sets the semaphore. The expressions from this theme cannot be modified. Global semaphores are not displayed.

## Processes

This theme lists the processes started since the beginning of the working session. The value column displays the time used and the current state for each process (i.e., Executing, Paused, and so on). The expressions from this theme cannot be modified.

## Tables and Fields

This theme lists the tables and fields in the 4D database. For each Table item, the Value column displays the size of the current selection for the current process as well as the number of **locked records**.

For each Field item, the Value column displays the value of the field for the current

record (except picture and BLOB). You can modify the field values but not the the tables' information.

## Sets

This theme lists the sets defined in the current process (the one you're currently tracing) and the interprocess sets. For each set, the Value column displays the number of records and the table name. The expressions from this theme cannot be modified.

## Named Selections

This theme lists the named selections that are defined in the current process (the one you're currently tracing); it also lists the interprocess named selections. For each named selection, the Value column displays the number of records and the table name. The expressions from this theme cannot be modified.

## Information

This theme contains general information regarding database operation, such as the current default table (if one exists), physical, virtual, free and used memory space, query destination, etc.

## Web

This theme displays information regarding the main Web server of the application (only available if the Web server is active):

- Web File To Send: name of Web file waiting to be sent (if any)
- Web Cache Usage: number of pages present in Web cache as well as its use percentage
- Web Server Elapsed Time: duration of Web server use in hours:minutes:seconds format
- Web Hits Count: total number of HTTP requests received since Web server launch, as well as the instantaneous number of requests per second
- Number of active Web processes: number of active Web processes, all Web processes together

The expressions contained within this theme cannot be modified.

## Contextual Menu

Additional options are available from the contextual menu of the Watch pane.

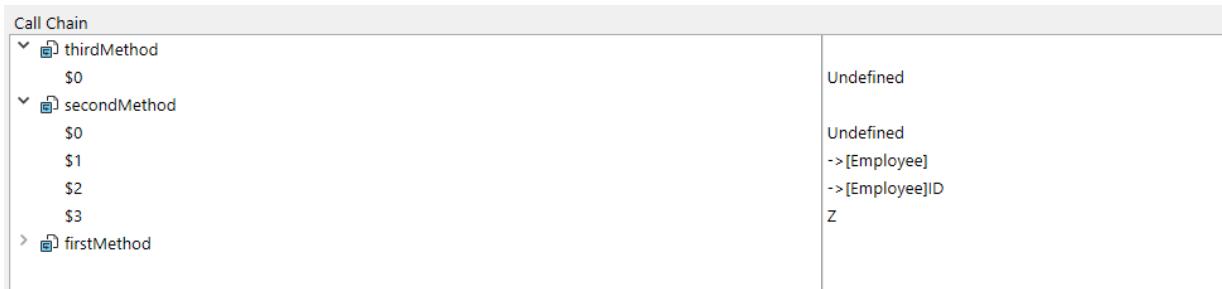
- **Collapse All:** Collapses all levels of the hierarchical list.
- **Expand All:** Expand all levels of the hierarchical list.
- **Show Types:** Displays the type of each item (when appropriate).
- **Show Field and Table Numbers:** Displays the number of each table or field. Useful if you work with table or field numbers, or with pointers using commands such as `Table` or `Field`.
- **Show Icons:** Displays an icon denoting the object type for each object. You can turn this option off in order to speed up the display, or just because you prefer to use only the **Show Types** option.
- **Sorted Tables and Fields:** Sorts the tables and fields in alphabetical order within their respective lists.
- **Show Integers in Hexadecimal:** Numbers are usually displayed in decimal

notation. This option displays them in hexadecimal notation. Note: To enter a numeric value in hexadecimal, type 0x (zero + "x"), followed by the hexadecimal digits.

- **Enable activity monitoring:** Activates the monitoring of activity (advanced checking of internal activity of the application) and displays the information retrieved in the additional themes: **Scheduler**, **Web** and **Network**.

## Call Chain Pane

A method may call other methods or class functions, which may call other methods or functions. The Call Chain pane lets you keep track of that hierarchy.



Each main level item is the name of a method or class function. The top item is the one you are currently tracing, the next main level item is the name of the caller (the method or function that called the one you are currently tracing), the next one is the caller's caller, and so on.

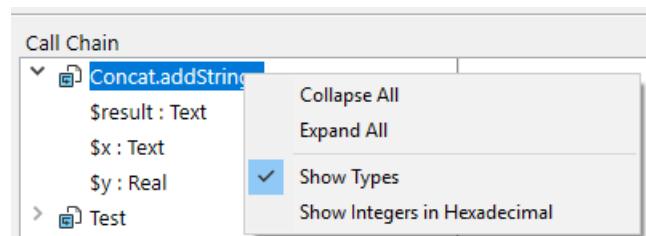
In the image above:

- `thirdMethod` has not received any parameter
- `$0` is currently undefined, as the method did not assign any value to `$0` (because it has not executed this assignment yet or because the method is a subroutine and not a function)
- `secondMethod` has received three parameters from `firstMethod`:
  - `$1` is a pointer to the `[Employee]` table
  - `$2` is a pointer to the `ID` field in the `[Employee]` table
  - `$3` is an alphanumeric parameter whose value is "Z"

You can double-click the name of any method to display its contents in the [Source Code Pane](#).

Clicking the icon next to a method or function name expands or collapses the parameters and the result (if any). Values appear on the right side of the pane. Clicking on any value on the right side allows you to change the value of any parameter or function result.

To display the parameter type, check the **Show types** option in the contextual menu:



After you deploy the list of parameters, you can drag and drop parameters and function results to the [Custom Watch Pane](#).

You can also use the [Get call chain](#) command to retrieve the call chain

programmatically.

## Custom Watch Pane

The Custom Watch Pane is useful for evaluating expressions. It is similar to the [Watch Pane](#), except here you decide which expressions are displayed. Any type of expression can be evaluated:

- field
- variable
- pointer
- calculation
- 4D command
- method
- and anything else that returns a value

Expression	Value
\$text	"Hello, World!"
\$calcResult	3
\$pField	->[Employee]ID
\$myBlob	10 Ko

You can evaluate any expression that can be shown in text form. This does not cover picture and BLOB fields or variables. To display BLOB contents, you can use BLOB commands, such as [BLOB to text](#).

### Handling expressions

There are several ways to add expressions to the list:

- Drag and drop an object or expression from the Watch Pane or the Call Chain Pane
- Select an expression in the [Source Code pane](#) and press **ctrl+D** (Windows) or **cmd+D** (macOS)
- Double-click somewhere in the empty space of the Custom Watch Pane (adds an expression with a placeholder name that you can edit)

You can enter any formula that returns a result.

To edit an expression, click on it to select it, then click again or press **Enter** on your keyboard.

To delete an expression, click on it to select it, then press **Backspace** or **Delete** on your keyboard.

**Warning:** Be careful when you evaluate a 4D expression modifying the value of one of the System Variables (for instance, the OK variable) because the execution of the rest of the method may be altered.

### Contextual Menu

The Custom Watch Pane's context menu gives you access the 4D formula editor and other options:

**New Expression:** This inserts a new expression and displays the 4D Formula Editor.

For more information on the Formula Editor, see the [4D Design Reference manual](#).

- **Insert Command:** Shortcut for inserting a 4D command as a new expression.
- **Delete All:** Removes all expressions from the Custom Watch Pane.
- **Standard Expressions:** Copies the Watch Pane's list of expressions.

This option is not available in remote debugging mode (see [Debugging from Remote Machines](#)).

- **Collapse All/Expand All:** Collapses or Expands all the hierarchical lists.
- **Show Types:** Displays the type of each item in the list (when appropriate).
- **Show Field and Table Numbers:** Displays the number of each table or field of the **Fields**. Useful if you work with tables, field numbers or pointers using the commands such as `Table` or `Field`.
- **Show Icons:** Displays an icon denoting the type of each item.
- **Sorted Tables and Fields:** Displays the table and fields in alphabetical order.
- **Show Integers in Hexadecimal:** Displays numbers using hexadecimal notation.  
To enter a numeric value in hexadecimal, type `0x` (zero + "x"), followed by the hexadecimal digits.

## Source Code Pane

The Source Code Pane shows the source code of the method or function currently being traced.

This area also allows you to add or remove [break points](#).

### Tool tip

Hover your pointer over any expression to display a tool tip that indicates:

- the declared type of the expression
- the current value of the expression

This also works with selections:

### Adding expressions to the Custom Watch Pane

You can copy any selected expression from the Source Code Pane to the [Custom Watch Pane](#).

1. In the Source code pane, select the expression to evaluate
2. Do one of the following:
  - Drag and drop the selected text to the Expression area of the Custom Watch Pane
  - Press **Ctrl+D** (Windows) or **Cmd+D** (macOS)
  - Right-click the selected text > **Copy to Expression Pane**

### Program Counter

The yellow arrow in the left margin of the Source Code pane is called the program counter. It marks the next line to be executed.

By default, the program counter line (also called the running line) is highlighted in the

debugger. You can customize the highlight color in the [Methods page of the Preferences](#).

## Moving the program counter

For debugging purposes, you can move the program counter for the method at the top of the call chain (the method currently executing). To do so, click and drag the yellow arrow to another line.

This only tells the debugger to pursue tracing or executing from a different point. It does not execute lines or cancel their execution. All current settings, fields, variables, etc. are not impacted.

For example:

```
// ...
If(This condition)
    DO_SOMETHING
Else
    DO_SOMETHING_ELSE
End if
// ...
```

Say the program counter is set to the line `If (This condition)`. When you click the **Step over** button, the program counter moves directly to the `DO_SOMETHING_ELSE` line. To examine the results of the `DO_SOMETHING` line, you can move the program counter to that line and execute it.

## Contextual menu

The contextual menu of the Source Code Pane provides access to several functions that are useful when executing methods in Trace mode:

- **Goto Definition:** Goes to where the selected object is defined. This command is available for:
  - *Project methods:* displays method contents in a new window of the Code Editor
  - *Fields:* Displays field properties in the inspector of the Structure window
  - *Tables:* Displays table properties in the inspector of the Structure window
  - *Forms:* Displays form in the Form editor
  - *Variables* (local, process, interprocess or \$n parameter): displays the line in the current method or among the compiler methods where the variable is declared
- **Search References** (also available in Code Editor): Searches all project objects (methods and forms) in which the current element of the method is referenced. The current element is the one selected or the one where the cursor is located. This can be the name of a field, variable, command, string, and so on. Search results are displayed in a new standard results window.
- **Copy:** Standard copy of the selected expression to the pasteboard.
- **Copy to Expression Pane:** Copy the selected expression to the Custom Watch Pane.
- **Run to Cursor:** Executes statements found between the program counter and the selected line of the method (where the cursor is found).
- **Set Next Statement:** Moves program counter to the selected line without executing this line or any intermediate ones. The designated line is only run if the user clicks on one of the execution buttons.
- **Toggle Breakpoint** (also available in Code Editor): Alternately inserts or removes

the breakpoint corresponding to the selected line. This modifies the breakpoint permanently: for instance, if you remove a breakpoint in the debugger, it no longer appears in the original method.

- **Edit Breakpoint** (also available in Code Editor): Displays the Breakpoint Properties dialog box. Any changes made modify the breakpoint permanently.

## Find Next/Previous

Specific shortcuts allow you to find strings identical to the one selected:

- To search for the next identical strings, press **Ctrl+E** (Windows) or **Cmd+E** (macOS)
- To search for the previous identical strings, press **Ctrl+Shift+E** (Windows) or **Cmd+Shift+E** (macOS)

The search is carried out only if you select at least one character in the Source code pane.

## Shortcuts

This section lists all the shortcuts available in the debugger window.

The tool bar also has [shortcuts](#).

### Watch Pane & Custom Watch Pane

- **Double-click** an item in the Watch Pane to copy it to the Custom Watch Pane
- **Double-Click** in the Custom Watch Pane to create a new expression

### Source Code Pane

- Click in the left margin to set or remove break points.
- **Alt+Shift+Click** (Windows) or **Option+Shift+Click** (macOS) sets a temporary break point.
- **Alt-Click** (Windows) or **Option-Click** displays the Edit Break window for a new or existing break point.
- A selected expression or object can be copied to the Custom Watch Pane by simple drag and drop.
- **Ctrl+D** (Windows) or **Cmd+D** (macOS) key combinations copy the selected text to the Custom Watch Pane.
- **Ctrl+E** (Windows) or **Cmd+E** (macOS) key combinations find the next strings identical to the one selected.
- **Ctrl+Shift+E** (Windows) or **Cmd+Shift+E** (macOS) key combinations find the previous strings identical to the one selected.

### All Panes

- **Ctrl + +/-** (Windows) or **Command + +/-** (macOS) increases or decreases the font size for a better readability. The modified font size is also applied to the Code Editor and is stored in the Preferences.
- **Ctrl + \*** (Windows) or **Command + \*** (macOS) forces the updating of the Watch Pane.
- When no item is selected in any pane, press **Enter** to step over.
- When an item value is selected, use the arrows keys to navigate through the list.
- When editing an item, use the arrow keys to move the cursor. Use **Ctrl-A/X/C/V** (Windows) or **Command-A/X/C/V** (macOS) as shortcuts to the Select All/Cut/Copy/Paste menu commands of the Edit menu.



# Breakpoints and Command Catching

## Overview

Breakpoints and command catching are very efficient debugging techniques. Both have the same effect: they pause the code execution (and display the debugger window if not already displayed) at a desired step.

You set breakpoints on any line of code where you want the execution to be paused. You can associate a condition to the break point.

Catching a command enables you to start tracing the execution of any process as soon as a command is called by that process.

## Breakpoints

To create a break point, click in the left margin of the Source Code pane in the debugger or in the Code Editor.

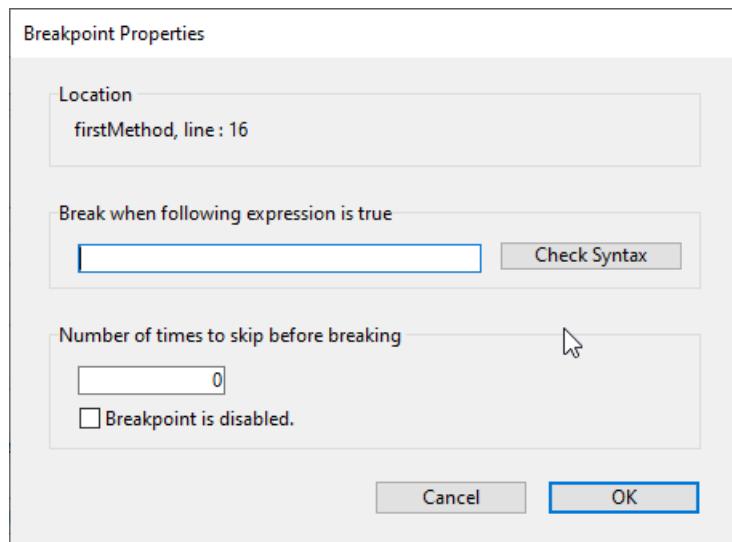
In the following example, a break point (the red bullet) has been set, in the debugger, on the line `If ($in.dataClass#Null)`:

In the above example, clicking the **No Trace** button resumes normal execution up to the line marked with the break point. That line is not executed itself — you are taken back to trace mode. Setting a break point beyond the program counter and clicking the **No Trace** button allows you to skip portions of the method being traced.

To remove a break point, click the corresponding bullet.

## Breakpoint Properties

You can edit the behavior of a breakpoint using the Breakpoint Properties window:



This window is available from the Code Editor or the [Source Code Pane](#). You can:

- right-click a line and select **Edit Breakpoint** in the contextual menu, or
- `Alt+click` (Windows) or `Option+click` (macOS) in the left margin.

If a break point already exists, the window is displayed for that break point. Otherwise, a break point is created and the window is displayed for the newly created break point.

Here is a description of the properties:

- **Location:** indicates the name of the method and the line number attached to the breakpoint.
- **Break when following expression is true:** You can create **conditional breakpoints** by entering a 4D formula that returns `True` or `False`. For example, insert `Records in selection(\[aTable]\)=0` to make sure the break occurs only if there no record selected for the table `[aTable]`. Breakpoint conditions are available in the **Condition** column of the [Break list](#).
- **Number of times to skip before breaking:** You can attach a breakpoint to a line located in a loop structure (While, Repeat, or For) or located in subroutine or function called from within a loop.
- **Breakpoint is disabled:** If you currently do not need a break point, but might need it later, you can temporarily disable it. A disabled break point appears as a dash (-) instead of a bullet (•)

## Breakpoints in remote debugging

The break point list is stored locally. In remote debugging mode, if the attached debugger is a remote 4D, the remote break point list replaces temporarily the server break point list during the debugging session.

The server break point list is automatically restored if it becomes again the attached debugger.

## Break List

The Break list is a page of the Runtime Explorer that lets you manage the breakpoints created in the Debugger Window or in the Code Editor. For more information on the Runtime Explorer, see its dedicated page in [the Design reference manual](#).

To open the Break list page:

1. From the **Run menu**, click **Runtime Explorer...**
2. Click the **Break** tab to display the Break list:

Using this window, you can:

- Set conditions for breakpoints in the **Conditions** column
- Enable or disable breakpoints by clicking the bullets in the margin. Disabled breakpoints display transparent bullets
- Delete breakpoints by pressing the `Delete` or `Backspace` key, or click on the **Delete** button below the list.
- Open the methods where the breakpoint are located by double-clicking any line in the list

You cannot add new breakpoints from this window. Breakpoints can only be created from within the Debugger window or the Code Editor.

## Catching Commands

The **Catch** tab of the Runtime Explorer lets you add additional breaks to your code by

catching calls to 4D commands. Unlike a break point, which is located in a particular project method (and therefore triggers a trace exception only when it is reached), the scope of catching a command includes all the processes that execute 4D code and call that command.

Catching a command is a convenient way to trace large portions of code without setting break points at arbitrary locations. For example, if a record that should not be deleted is deleted after you've executed one or several processes, you can try to reduce the field of your investigation by catching commands such as `DELETE RECORD` and `DELETE SELECTION`. Each time these commands are called, you can check if the record in question has been deleted, and thus isolate the faulty part of the code.

Feel free to combine breakpoints and command catching.

To open the Caught Commands page:

1. Choose **Run > Runtime explorer...** to open the Runtime Explorer.
2. Click **Catch** to display the Caught Commands List:

This page lists the commands to be caught during execution. It is composed of two columns:

- The left column displays the Enable/Disable status of the caught command, followed by the name of the command
- The right column displays the condition associated with the caught command, if any

To add a command to be caught:

1. Click on the **Add New Catch** button (in the shape of a +) located below the list. A new entry is added to the list with the `ALERT` command as default
2. Click the **ALERT** label, type the name of the command you want to catch, then press **Enter**.

To enable or disable a caught command, click on the bullet (•) in front of the command label. The bullet is transparent when the command is disabled.

Disabling a caught command has almost the same effect as deleting it. During execution, the debugger spends almost no time on the entry. The advantage of disabling an entry is that you do not have to recreate it when you need it again.

To delete a caught command:

1. Select a command in the list.
2. Press **Backspace** or **Delete** on your keyboard or click on the **Delete** button beneath the list (**Delete All** removes all commands in the list).

## Setting a Condition for catching a command

1. Click on the entry in the right column
2. Enter a 4D formula (expression, command call or project method) that returns a Boolean value.

To remove a condition, delete its formula.

Adding conditions allows you to stop execution when the command is invoked only if

the condition is met. For example, if you associate the condition `Records in selection(\[Emp]>10)` with the break point on the `DELETE SELECTION` command, the code will not be stopped during execution of the `DELETE SELECTION` command if the current selection of the [Emp] table only contains 9 records (or less).

Adding conditions to caught commands slows the execution, because the condition has to be evaluated each time an exception is met. On the other hand, adding conditions accelerates the debugging process, because 4D automatically skips occurrences that do not match the conditions.

# Debugging from remote machines

## Overview

When a 4D database is running on 4D Server in interpreted mode, you can debug the 4D code running on the server from a remote 4D client logged to the project. You just need to attach the debugger to a specific remote machine, and the code execution can be monitored in the debugger directly on the remote machine.

On a remote machine, the [debugger window](#) displays a specific server icon and a blue background color to indicate that you are debugging server code:

This feature is particularly useful when 4D Server runs in headless mode (see [Command Line Interface](#)), or when access to the server machine is not easy.

## Attached debugger

Only one debugger can debug a 4D Server application at a given time. It is called the **attached debugger**. The attached debugger can be:

- the local 4D Server debugger (default) - if the server is not running headless.
- the debugger of a remote 4D client - if the remote session has access to Design mode.

The attached debugger is called whenever a 4D Server encounters:

- a break point
- a `TRACE` command
- a caught command
- an error

Keep in mind that error messages are sent to the attached debugger machine. This means that in the case of a remote debugger, server error messages are displayed on the remote 4D client.

Note that:

- The code executed in the `On Server Startup Database` Method cannot be debugged remotely. It can only be debugged on the server side
- If no debugger is attached, the running code is not stopped by debugging commands

## Attaching the debugger

By default when you start an interpreted application:

- if 4D Server is not running headless, the debugger is attached to the server,
- if 4D Server is running headless, no debugger is attached.

You can attach the debugger to any remote 4D client allowed to connect to the 4D Server application.

The remote 4D client's user session must have access to the Design environment of the database.

To attach the debugger to a remote 4D client:

1. In the 4D Server menu bar, select **Edit > Detach Debugger** so that the debugger becomes available to remote machines (this step is useless if the 4D Server is running headless).
2. In a remote 4D client connected to the server, select **Run > Attach Remote Debugger**

If the attachment is accepted (see [Rejected attachment requests](#)), the menu command becomes **Detach Remote Debugger**.

The debugger is then attached to the remote 4D client:

- until the end of the user session
- until you select `Detach Remote Debugger`

To attach the debugger back to the server:

1. On the remote 4D client that has the debugger attached, select **Run > Detach Remote Debugger**.
2. In the 4D Server menu bar, select **Edit > Attach debugger**.

When the debugger is attached to the server (default), all server processes are automatically executed in cooperative mode to enable debugging. This can have a significant impact on performance. When you don't need to debug on the server machine, it is recommended to detach the debugger and attach it to a remote machine if necessary.

## Attaching debugger at startup

4D allows you to automatically attach the debugger to a remote 4D client or the server at startup:

- On the server (if not headless), this option is named **Attach Debugger At Startup**. When the server is started, it automatically attaches the debugger (default).  
**Warning:** If this option is selected for a server which is subsequently launched in headless mode, the debugger won't be available for this server.
- On a remote 4D client, this option is named **Attach Remote Debugger At Startup**. When selected, the remote 4D client will automatically try to attach the remote debugger at each subsequent connection to the same 4D Server database. If the attachment is accepted (see [Rejected attachment requests](#)), the remote debugger is automatically attached to the remote 4D client and the **Detach Remote Debugger option is displayed**.

This setting is applied per project and is stored locally in the `.4DPreferences` file.

## Rejected attachment requests

While the debugger is already attached to a remote 4D client or to 4D Server, no other machine can attach the debugger.

If a machine tries to attach the debugger while it is already attached, the attachment is rejected and a dialog box appears:

4D Server



Unable to attach debugger. Debugger is already attached to a remote 4D:  
- 4D User: Designer  
- Machine name: OPT-990-1043  
- Session name: aschmitt

OK

4D



Unable to attach debugger. Debugger is already attached to the server.

OK

Attaching the debugger in this case requires that:

- the attached debugger is detached from the server or from the remote 4D client using respectively the **Detach debugger** or **Detach remote debugger** menu command,
- the attached remote 4D client session is closed.

## Description of log files

4D applications can generate several log files that are useful for debugging or optimizing their execution. Logs are usually started or stopped using selectors of the [SET DATABASE PARAMETER](#) or [WEB SET OPTION](#) commands and are stored in the [Logs folder](#) of the project.

Information logged needs to be analyzed to detect and fix issues. This section provides a comprehensive description of the following log files:

- [4DRequestsLog.txt](#)
- [4DRequestsLog\\_ProcessInfo.txt](#)
- [HTTPDebugLog.txt](#)
- 4DDebugLog.txt ([standard](#) & [tabular](#))
- [4DDiagnosticLog.txt](#)
- [4DIMAPLog.txt](#)
- [4DPOP3Log.txt](#)
- [4DSMTPLog.txt](#)
- [ORDA requests log file](#)

When a log file can be generated either on 4D Server or on the remote client, the word "Server" is added to the server-side log file name, for example "4DRequestsLogServer.txt"

Log files share some fields so that you can establish a chronology and make connections between entries while debugging:

- `sequence_number`: this number is unique over all debug logs and is incremented for each new entry whatever the log file, so that you can know the exact sequence of the operations.
- `connection_uuid`: for any 4D process created on a 4D client that connects to a server, this connection UUID is logged on both server and client side. It allows you to easily identify the remote client that launched each process.

## 4DRequestsLog.txt

This log file records standard requests carried out by the 4D Server machine or the 4D remote machine that executed the command (excluding Web requests).

How to start this log:

- on the server:

```
SET DATABASE PARAMETER(4D Server log recording;1)
//server side
```

- on a client:

```
SET DATABASE PARAMETER(Client Log Recording;1)
//remote side
```

This statement also starts the [4DRequestsLog\\_ProcessInfo.txt](#) log file.

## Headers

This file starts with the following headers:

- Log Session Identifier
- Hostname of the server that hosts the application
- User Login Name: login on the OS of the user that ran the 4D application on the server.

## Contents

For each request, the following fields are logged:

Field name	Description
sequence_number	Unique and sequential operation number in the logging session
time	Date and time using ISO 8601 format: 'YYYY-MM-DDTHH:MM:SS.mmm'
systemid	System ID
component	Component signature (e.g., '4SQLS' or 'dbmg')
process_info_index	Corresponds to the "index" field in 4DRequestsLog_ProcessInfo.txt log, and permits linking a request to a process.
request	<a href="#">C/S or ORDA request ID</a> or message string for SQL requests or <a href="#">LOG EVENT</a> messages
bytes_in	Number of bytes received
bytes_out	Number of bytes sent
server_duration   exec_duration	Depends on where the log is generated: <ul style="list-style-type: none"> <li>• <i>server_duration</i> when generated on the client --Time taken in microseconds for the server to process the request and return a response. B to F in image below, OR</li> <li>• <i>exec_duration</i> when generated on the server --Time taken in microseconds for the server to process the request. B to E in image below.</li> </ul>
write_duration	Time taken in microseconds for sending the: <ul style="list-style-type: none"> <li>• Request (when run on the client). A to B in image below.</li> <li>• Response (when run on the server). E to F in image below.</li> </ul>
task_kind	Preemptive or cooperative (respectively 'p' or 'c')
rtt	Time estimate in microseconds for the client to send the request and the server to acknowledge it. A to D and E to H in image below. <ul style="list-style-type: none"> <li>• Only measured when using the ServerNet network layer, returns 0 when used with the legacy network layer.</li> <li>• For Windows versions prior to Windows 10 or Windows Server 2016, the call will return 0.</li> </ul>
extra	Additional information related to the context, for example dataclass name and/or attribute name in case of ORDA request

Request flow:

## 4DRequestsLog\_ProcessInfo.txt

This log file records information on each process created on the 4D Server machine or the 4D remote machine that executed the command (excluding Web requests).

How to start this log:

- on the server:

```
SET DATABASE PARAMETER(4D Server log recording;1) //server side
```

- on a client:

```
SET DATABASE PARAMETER(Client Log Recording;1) //remote side
```

This statement also starts the [4DRequestsLog.txt](#) log file.

## Headers

This file starts with the following headers:

- Log Session Identifier
- Hostname of the server that hosts the application
- User Login Name: login on the OS of the user that ran the 4D application on the server.

## Contents

For each process, the following fields are logged:

Field name	Description
sequence_number	Unique and sequential operation number in the logging session
time	Date and time using ISO 8601 format: "YYYY-MM-DDTHH:MM:SS.mmm"
process_info_index	Unique and sequential process number
CDB4DBaseContext	DB4D component database context UUID
systemid	System ID
server_process_id	Process ID on Server
remote_process_id	Process ID on Client
process_name	Process name
cID	Identifier of 4D Connection
uID	Identifier of 4D Client
IP Client	IPv4/IPv6 address
host_name	Client hostname
user_name	User Login Name on client
connection_uuid	UUID identifier of process connection
server_process_unique_id	Unique process ID on Server

## HTTPDebugLog.txt

This log file records each HTTP request and each response in raw mode. Whole requests, including headers, are logged; optionally, body parts can be logged as well.

How to start this log:

```
WEB SET OPTION(Web debug log;wdl enable without body)
//other values are available
```

The following fields are logged for both Request and Response:

<b>Field name</b>	<b>Description</b>
SocketID	ID of socket used for communication
PeerIP	IPv4 address of host (client)
PeerPort	Port used by host (client)
TimeStamp	Timestamp in milliseconds (since system startup)
ConnectionID	Connection UUID (UUID of VTCPSocket used for communication)
SequenceNumber	Unique and sequential operation number in the logging session

## 4DDebugLog.txt (standard)

This log file records each event occurring at the 4D programming level. Standard mode provides a basic view of events.

How to start this log:

```
SET DATABASE PARAMETER(Debug Log Recording;2)
//standard, all processes
```

```
SET DATABASE PARAMETER(Current process debug log recording;2)
//standard, current process only
```

The following fields are logged for each event:

<b>Column #</b>	<b>Description</b>
1	Unique and sequential operation number in the logging session
2	Date and time in ISO 8601 format (YYYY-MM-DDThh:mm:ss.mmm)
3	Process ID (p=xx) and unique process ID (puid=xx)
4	Stack level
5	Can be Command Name/ Method Name/Message/ Task Start Stop info/Plugin Name, event or Callback/Connection UUID
6	Time taken for logging operation in milliseconds

## 4DDebugLog.txt (tabular)

This log file records each event occurring at the 4D programming level in a tabbed, compact format that includes additional information (compared to the standard format).

How to start this log:

```
SET DATABASE PARAMETER(Debug Log Recording;2+4)
//extended tabbed format, all processes
```

```
SET DATABASE PARAMETER(Current process debug log recording;2+4)
//extended, current process only
```

The following fields are logged for each event:

<b>Column #</b>	<b>Field name</b>	<b>Description</b>
1	sequence_number	Unique and sequential operation number in the logging session
2	time	Date and time in ISO 8601 format (YYYY-MM-DDThh:mm:ss.mmm)
3	ProcessID	Process ID
4	unique_processID	Unique process ID

Column #	Field name	Description
6	operation_type	<p>Stack level Log operation type. This value may be an absolute value:</p> <ol style="list-style-type: none"> <li>1. Command</li> <li>2. Method (project method, database method, etc.)</li> <li>3. Message (sent by <a href="#">LOG EVENT</a> command only)</li> <li>4. PluginMessage</li> <li>5. PluginEvent</li> <li>6. PluginCommand</li> <li>7. PluginCallback</li> <li>8. Task</li> <li>9. Member method (method attached to a collection or an object)</li> </ol> <p>When closing a stack level, the <code>operation_type</code>, <code>operation</code> and <code>operation_parameters</code> columns have the same value as the opening stack level logged in the <code>stack_opening_sequence_number</code> column. For example:</p> <ol style="list-style-type: none"> <li>1. 121 15:16:50:777 5 8 1 2 CallMethod Parameters 0</li> <li>2. 122 15:16:50:777 5 8 2 1 283 0</li> <li>3. 123 15:16:50:777 5 8 2 1 283 0 122 3</li> <li>4. 124 15:16:50:777 5 8 1 2 CallMethod Parameters 0 121 61</li> </ol> <p>The 1st and 2nd lines open a stack level, the 3rd and 4th lines close a stack level. Values in the columns 6, 7 and 8 are repeated in the closing stack level line. The column 10 contains the stack level opening sequence numbers, i.e. 122 for the 3rd line and 121 for the 4th.</p> <p>May represent (depending on operation type):</p> <ul style="list-style-type: none"> <li>• a Language Command ID (when type=1)</li> <li>• a Method Name (when type=2)</li> <li>• a combination of pluginIndex;pluginCommand (when type=4, 5, 6 or 7). May contain something like '3;2'</li> <li>• a Task Connection UUID (when type=8)</li> </ul> <p>Parameters passed to commands, methods, or plugins</p> <p>Form event if any; empty in other cases (suppose that column is used when code is executed in a form method or object method)</p> <p>Only for the closing stack levels: Sequence number of the corresponding opening stack level</p>
7	operation	
8	operation_parameters	
9	form_event	
10	stack_opening_sequence_number	

Column #	Field name	Description
11	stack_level_execution_time	Only for the closing stack levels: Elapsed time in micro seconds of the current logged action; only for the closing stack levels (see 10th columns in lines 123 and 124 in the log above)

## 4DDiagnosticLog.txt

This log file records many events related to the internal application operation and is human-readable. You can include custom information in this file using the [LOG EVENT](#) command.

How to start this log:

```
SET DATABASE PARAMETER(Diagnostic log recording;1) //start recording
```

The following fields are logged for each event:

Field Name	Description
sequenceNumber	Unique and sequential operation number in the logging session
timestamp	Date and time in ISO 8601 format (YYYY-MM-DDThh:mm:ss.mmm)
loggerID	Optional
componentSignature	Optional - internal component signature
messageLevel	Trace, Debug, Info, Warning, Error
message	Description of the log entry

Depending on the event, various other fields can also be logged, such as task, socket, etc.

### Diagnostic log levels

The *4DDiagnosticLog.txt* file can log different levels of messages, from `ERROR` (most important) to `TRACE` (less important). By default, the `INFO` level is set, which means that the file will log only important events, including errors and unexpected results (see below).

You can select the level of messages using the `Diagnostic log level` selector of the [SET DATABASE PARAMETER](#) command, depending on your needs. When you select a level, levels above (which are more important) are implicitly selected also. The following levels are available:

Message level	Description	When selected, includes
ERROR	A part of the application does not work	ERROR
WARN	Potential error, use of a deprecated function, poor uses, undesirable or unexpected situation	ERROR, WARN
INFO	<i>Default level</i> - Important application event	ERROR, WARN, INFO
DEBUG	Detail of application flow (for 4D technical services)	ERROR, WARN, INFO, DEBUG
TRACE	Other internal information (for 4D technical services)	ERROR, WARN, INFO, DEBUG, TRACE

## 4DSMTPLLog.txt, 4DPOP3Log.txt, and 4DIMAPLog.txt

These log files record each exchange between the 4D application and the mail server (SMTP, POP3, IMAP) that has been initiated by the following commands:

- SMTP - [SMTP New transporter](#)
- POP3 - [POP3 New transporter](#)
- IMAP - [IMAP New transporter](#)

The log files can be produced in two versions:

- a regular version:
  - named 4DSMTPLog.txt, 4DPOP3Log.txt, or 4DIMAPLog.txt
  - no attachments
  - uses an automatic circular file recycling each 10 MB
  - intended for usual debugging

To start this log:

```
SET DATABASE PARAMETER(SMTP Log;1) //start SMTP log
SET DATABASE PARAMETER(POP3 Log;1) //start POP3 log
SET DATABASE PARAMETER(IMAP Log;1) //start IMAP log
```

**4D Server:** Click on the **Start Request and Debug Logs** button in the [Maintenance Page](#) of the 4D Server administration window.

This log path is returned by the `Get 4D file` command.

- an extended version:
  - attachment(s) included no automatic recycling
  - custom name
  - reserved for specific purposes

To start this log:

```
$server:=New object
...
//SMTP
$server.logFile:="MySMTPAuthLog.txt"
$transporter:=SMTP New transporter($server)

// POP3
$server.logFile:="MyPOP3AuthLog.txt"
$transporter:=POP3 New transporter($server)

//IMAP
$server.logFile:="MyIMAPAuthLog.txt"
$transporter:=IMAP New transporter($server)
```

## Contents

For each request, the following fields are logged:

<b>Column #</b>	<b>Description</b>
1	Unique and sequential operation number in the logging session
2	Date and time in RFC3339 format (yyyy-mm-ddThh:mm:ss.ms)
3	4D Process ID
4	Unique process ID
5	<ul style="list-style-type: none"> <li>• SMTP,POP3, or IMAP session startup information, including server host name, TCP port number used to connect to SMTP,POP3, or IMAP server and TLS status,or</li> <li>• data exchanged between server and client, starting with "S &lt;" (data received from the SMTP,POP3, or IMAP server) or "C &gt;" (data sent by the SMTP,POP3, or IMAP client): authentication mode list sent by the server and selected authentication mode, any error reported by the SMTP,POP3, or IMAP Server, header information of sent mail (standard version only) and if the mail is saved on the server,or</li> <li>• SMTP,POP3, or IMAP session closing information.</li> </ul>

## ORDA requests

ORDA requests logs can record each ORDA request and server response. Two ORDA requests logs are available:

- a client-side ORDA request log, in .txt format
- a server-side ORDA request log, in .jsonl format

### Client-side

The client-side ORDA log records each ORDA request sent from a remote machine. You can direct log information to memory or to a .txt file on disk of the remote machine. The name and location of this log file are your choice.

How to start this log:

```
//on a remote machine
SET DATABASE PARAMETER(Client Log Recording;1)
ds.startRequestLog(File("/PACKAGE/Logs/ordaLog.txt"))
    //can be also sent to memory
SET DATABASE PARAMETER(Client Log Recording;0)
```

**(i) NOTE**

Triggering the client-side [4DRequestsLog.txt](#) using `SET DATABASE PARAMETER` is not mandatory. However, it is required if you want to log the unique `sequenceNumber` field.

The following fields are logged for each request:

Field name	Description	Example
sequenceNumber	Unique and sequential operation number in the logging session	104
url	Request URL	"rest/Persons(30001)"
startTime	Starting date and time using ISO 8601 format	"2019-05-28T08:25:12.346Z"
endTime	Ending date and time using ISO 8601 format	"2019-05-28T08:25:12.371Z"
duration	Client processing duration in milliseconds (ms)	25
response	Server response object	{"status":200,"body": {"__entityModel":"Persons", [...]}}

## Example

Here is an example of a client-side ORDA log file record:

```
{
    "sequenceNumber": 7880,
    "url": "rest/Employees/$entityset/F910C2E4A2EE6B43BBEE74A0A4F68E5A/Salary?
$compute='sum'&$progress4Dinfo='D0706F1E77D4F24985BE4DDE9FFA1739'",
    "startTime": "2023-05-15T10:43:39.400Z",
    "endTime": "2023-05-15T10:43:39.419Z",
    "duration": 19,
    "response": {
        "status": 200,
        "body": 75651
    }
}
```

## Server-side

The server-side ORDA log records each ORDA request processed by the server, as well as the server response (optional). Log information is saved in a .jsonl file on the server machine disk (by default, *ordaRequests.jsonl*).

How to start this log:

```
//on the server
SET DATABASE PARAMETER(4D Server log recording;1)
ds.startRequestLog(File("/PACKAGE/Logs/ordaRequests.jsonl");srl log
response without body)
    //srl... parameter is optional
SET DATABASE PARAMETER(4D Server log recording;0)
```

### **(i) NOTE**

Triggering the server-side [4DRequestsLog.txt](#) using `SET DATABASE PARAMETER` is not mandatory. However, it is required if you want to log the unique `sequenceNumber` and the `duration` fields.

The following fields are logged for each request:

Field name	Description	Example
sequenceNumber	Unique and sequential operation number in the logging session	104
url	Request URL	"rest/Persons(30001)"
startTime	Starting date and time using ISO 8601 format	"2019-05-28T08:25:12.346Z"
duration	Server processing duration in microseconds ( $\mu$ )	2500
response	Server response object, can be configured in <a href="#">.startRequestLog()</a>	{"status":200,"body": {"__entityModel":"Persons", [...]}}
ipAddress	User IP address	"192.168.1.5"
userName	Name of the 4D user	"henry"
systemUserName	Login name of the user on the machine	"hsmith"
machineName	Name of the user machine	"PC of Henry Smith"

## Example

Here is an example of a server-side ORDA log record:

```
{
  "url": "rest/Employees/$entityset/F910C2E4A2EE6B43BBEE74A0A4F68E5A/Salary?
$compute='sum'&$progress4Dinfo='D0706F1E77D4F24985BE4DDE9FFA1739'",
  "systemUserName": "Admin",
  "userName": "Designer",
  "machineName": "DESKTOP-QSK9738",
  "taskID": 5,
  "taskName": "P_1",
  "startTime": "2023-05-15T11:43:39.401",
  "response": {
    "status": 200,
    "body": 75651
  },
  "sequenceNumber": 7008,
  "duration": 240
}
```

## Using a log configuration file

You can use a **log configuration file** to easily manage log recording in a production environment. This file is preconfigured by the developer. Typically, it can be sent to customers so that they just need to select it or copy it in a local folder. Once enabled, the log configuration file triggers the recording of specific logs.

### How to enable the file

There are several ways to enable the log configuration file, depending on your configuration:

- **4D Server with interface:** you can open the Maintenance page and click on the [Load logs configuration file](#) button, then select the file. In this case, you can use any name for the configuration file. It is immediately enabled on the server.
- **an interpreted or compiled project:** the file must be named `logConfig.json` and copied in the [Settings folder](#) of the project (located at the same level as the [Project folder](#)). It is enabled at project startup (only on the server in

client/server).

- **a built application:** the file must be named `logConfig.json` and copied in the following folder:
  - Windows: `Users\[userName]\AppData\Roaming\[application]`
  - macOS: `/Users/[userName]/Library/ApplicationSupport/[application]`
- **all projects with a stand-alone or remote 4D:** the file must be named `logConfig.json` and copied in the following folder:
  - Windows: `Users\[userName]\AppData\Roaming\4D`
  - macOS: `/Users/[userName]/Library/ApplicationSupport/4D`
- **all projects with 4D Server:** the file must be named `logConfig.json` and copied in the following folder:
  - Windows: `Users\[userName]\AppData\Roaming\4D Server`
  - macOS: `/Users/[userName]/Library/ApplicationSupport/4D Server`

## NOTE

If a `logConfig.json` file is installed in both Settings and AppData/Library folders, the Settings folder file will have priority.

## JSON file description

The log configuration file is a `.json` file that must comply with the following json schema:

```
{  
    "$schema": "http://json-schema.org/draft-07/schema",  
    "title": "Logs Configuration File",  
    "description": "A file that controls the state of different types  
of logs in 4D clients and servers",  
    "type": "object",  
    "properties": {  
        "forceConfiguration": {  
            "description": "Forcing the logs configuration described  
in the file ingoring changes coming from code or user interface",  
            "type": "boolean",  
            "default": true  
        },  
        "requestLogs": {  
            "description": "Configuration for request logs",  
            "type": "object",  
            "properties": {  
                "clientState": {  
                    "description": "Enable/Disable client request logs  
(from 0 to N)",  
                    "type": "integer",  
                    "minimum": 0  
                },  
                "serverState": {  
                    "description": "Enable/Disable server request logs  
(from 0 to N)",  
                    "type": "integer",  
                    "minimum": 0  
                }  
            }  
        },  
        "debugLogs": {  
            "description": "Configuration for debug logs",  
            "type": "object",  
            "properties": {  
                "commandList": {  
                    "type": "array",  
                    "items": "\"commandName\" : \"commandValue\""  
                }  
            }  
        }  
    }  
}
```

```

        "description": "Commands to log or not log",
        "type": "array",
        "items": {
            "type": "string"
        },
        "minItems": 1,
        "uniqueItems": true
    },
    "state": {
        "description": "integer to specify type of
debuglog and options",
        "type": "integer",
        "minimum": 0
    }
}
},
"diagnosticLogs": {
    "description": "Configuration for debug logs",
    "type": "object",
    "properties": {
        "state": {
            "description": "Enable/Disable diagnostic logs 0
or 1 (0 = do not record, 1 = record)",
            "type": "integer",
            "minimum": 0
        },
        "level": {
            "description": "Configure diagnostic logs",
            "type": "integer",
            "minimum": 2,
            "maximum": 6
        }
    }
},
"httpDebugLogs": {
    "description": "Configuration for http debug logs",
    "type": "object",
    "properties": {
        "level": {
            "description": "Configure http request logs",
            "type": "integer",
            "minimum": 0,
            "maximum": 7
        },
        "state": {
            "description": "Enable/Disable recording of web
requests",
            "type": "integer",
            "minimum": 0,
            "maximum": 4
        }
    }
},
"POP3Logs": {
    "description": "Configuration for POP3 logs",
    "type": "object",
    "properties": {
        "state": {
            "description": "Enable/Disable POP3 logs (from 0
to N)",
            "type": "integer",
            "minimum": 0
        }
    }
}.
}

```

```

        "SMTPLogs": {
            "description": "Configuration for SMTP logs",
            "type": "object",
            "properties": {
                "state": {
                    "description": "Enable/Disable SMTP log recording
(form 0 to N)",
                    "type": "integer",
                    "minimum": 0
                }
            }
        },
        "IMAPLogs": {
            "description": "Configuration for IMAP logs",
            "type": "object",
            "properties": {
                "state": {
                    "description": "Enable/Disable IMAP log recording
(form 0 to N)",
                    "type": "integer"
                }
            }
        },
        "ORDALogs": {
            "description": "Configuration for ORDA logs",
            "type": "object",
            "properties": {
                "state": {
                    "description": "Enable/Disable ORDA logs (0 or
1)",
                    "type": "integer"
                },
                "filename": {
                    "type": "string"
                }
            }
        }
    }
}

```

## Example

Here is an example of log configuration file:

```
{  
  "forceLoggingConfiguration": false,  
  "requestLogs": {  
    "clientState": 1,  
    "serverState": 1  
  },  
  "debugLogs": {  
    "commandList": ["322", "311", "112"],  
    "state": 4  
  },  
  "diagnosticLogs": {  
    "state": 1  
  },  
  "httpDebugLogs": {  
    "level": 5,  
    "state": 1  
  },  
  "POP3Logs": {  
    "state": 1  
  },  
  "SMTPLogs": {  
    "state": 1  
  },  
  "IMAPLogs": {  
    "state": 1  
  },  
  "ORDALogs": {  
    "state": 1,  
    "filename": "ORDALog.txt"  
  }  
}
```

## Settings

The Settings configure how the current project functions. These parameters may be different for each project. They include the listening ports, backup configurations, security options, Web parameters, etc.

4D provides another set of parameters, called **Preferences**, that apply to the 4D IDE application. For more information, refer to [Preferences](#).

## Accessing the settings

You can access the Settings dialog box:

- using the **Design > Settings...** menu option
- by clicking **Settings** on the 4D toolbar
- on 4D Server, using the **Edit > Settings...** menu option

When [User settings mode is enabled](#), **Settings...** is renamed **Structure Settings...** and two additional menu commands are available at each location:

- **User Settings...** gives you access to settings that can be stored externally in a user file. If these are modified, they are used instead of structure settings.
- **User Settings for Data File...** gives you access to settings that can be stored externally in a user file attached to the current data file. If they are modified, they are used instead of user or structure settings.

## Locking information

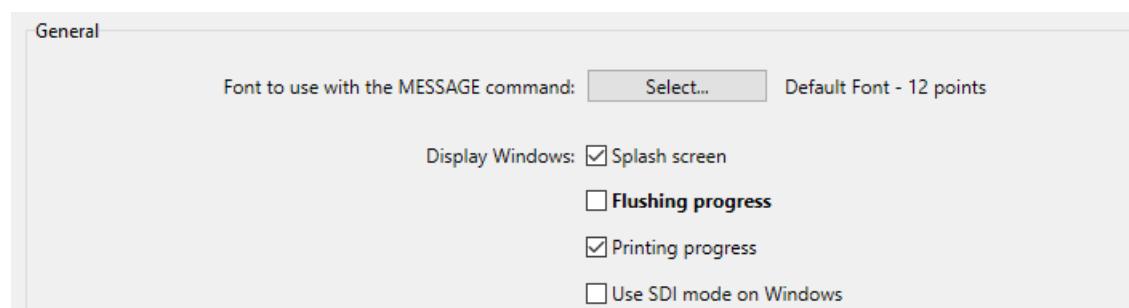
Locking can occur in both Project and Client/server modes when:

- The *settings.4DSettings* file is 'Read-only' (Projects only). Modifying a setting will display an alert to unlock it, if possible.
- Two or more users attempt to modify the same settings at the same time. The settings cannot be used until the first user frees it by closing the window. (Client/server only)

In both cases, the settings can be opened in 'Read-only', but cannot be used until the lock is removed.

## Customizing parameters

In the Settings dialog boxes, parameters whose values have been modified appear **in bold**:



Parameters indicated as customized may have been modified directly in the dialog

box, or may have been modified previously in the case of a converted project.

A parameter still appears in bold even when its value is replaced manually with its default values. This way it is always possible to visually identify any parameters that have been customized.

Most of the settings are applied immediately. However, a few of them (such as the Startup environment setting) only take effect when the database is restarted. In this case, a dialog box appears to inform you that the change will take effect at the next startup.

## Resetting the settings

To reset the parameters to their default values and remove the bold style indicating that they have been customized, click **Reset to factory settings**.

This button resets all the parameters of the current page. It becomes active when at least one parameter has been modified on the current page.

## General page

The General page contains options to configure generic parameters for the 4D project.

### Design

This area contains the **Display toolbar** option. When it is checked, the 4D toolbar is displayed in the Design environment.

### General

You use this area to set options concerning project startup and operation.

#### Startup Environment

You use this menu to select the default startup mode for the database: **Design** or **Application**. Unless specified, 4D opens by default in the Design environment if a password access system is not activated.

### Component

This area allows [component developers](#) to configure how their component classes and functions will be exposed in the 4D Code Editor once the component is installed.

#### Component namespace in the class store

Use this area to declare a namespace for the component classes and functions in the code on host projects. See [Declaring the component namespace](#).

#### Generate syntax file for code completion when compiled

When you check this option, a syntax file (JSON format) is automatically created during the compilation phase. See [Code completion for compiled components](#).

## Interface page

You use the Interface page to set various options related to the project interface.

### General

This area lets you set various options concerning display.

#### Font to use with the MESSAGE command

Click **Select...** to set the font and size for the characters used by the `MESSAGE` command.

The default font and its size depend on the platform where 4D is running.

This property also affects the following parts of 4D:

- certain preview areas of the Explorer
- the ruler of the Form editor

### Display Windows

Other options configure the display of various windows in the Application mode.

- **Splash screen:** When this option is deselected, the [splash screen of the current menu bar](#) does not appear in the Application mode. When you hide this window, it is up to you to manage the display of all your windows by programming, for example in the `On Startup` database method.
- **Flushing progress:** When this option is checked, 4D displays a window at the bottom left of the screen while the data in the cache is flushed. Since this operation momentarily blocks user actions, displaying this window lets them know that flushing is underway.

#### NOTE

You can set the [frequency for cache flushing](#) in **Settings > Database > Memory**.

- **Printing progress:** Lets you enable or disable the display of the printing progress dialog box when printing.
- **Use SDI mode on Windows:** When this option checked, 4D enables automatically the [SDI mode \(Single-Document Interface\)](#) in your application when executed in a [supported context](#). When you select this option, on Windows the **Run** menu of the 4D menu bar allows you to select the mode in which you want to test the application:

#### NOTE

This option can be selected on macOS but will be ignored when the application is executed on this platform.

### Appearance

This menu lets you select the color scheme to use at the main application level. A color scheme defines a global set of interface colors for texts, backgrounds, windows, etc., used in your forms.

This option only works on macOS. On Windows, the "Light" scheme is always used.

The following schemes are available:

- **Light**: the application will use the Default Light Theme
- **Dark**: the application will use the Default Dark Theme
- **Inherited** (default): the application will inherit from the higher priority level (i.e., OS user preferences)

Default themes can be handled using CSS. For more information, please refer to the [Media Queries](#) section.

The main application scheme will be applied to forms by default. However, it can be overridden:

- by the [SET APPLICATION COLOR SCHEME](#) command at the working session level;
- using by the [Color Scheme](#) form property at each form level (highest priority level). **Note:** When printed, forms always use the "Light" scheme.

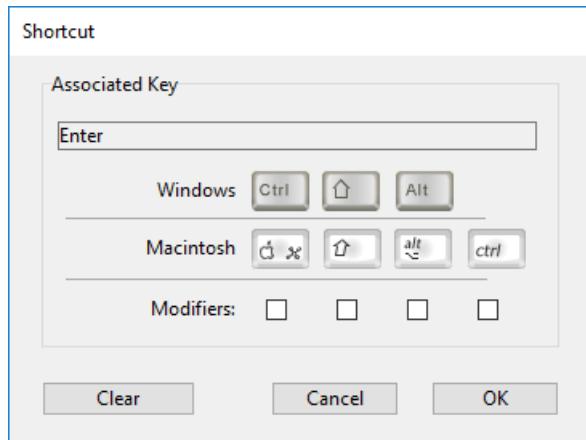
## Shortcuts

You use the Shortcuts area for viewing and modifying default shortcuts for three basic 4D form operations in your desktop applications. These shortcuts are identical for both platforms. Key icons indicate the corresponding Windows and macOS keys.

The default shortcuts are as follows:

- Accept input form: **Enter**
- Cancel input form: **Esc**
- Add to subform: **Ctrl+Shift+{/}** (Windows) or **Command+Shift+{/}** (macOS)

To change the shortcut of an operation, click the corresponding **Edit** button. The following dialog box appears:



To change the shortcut, type the new key combination on your keyboard and click **OK**. If you prefer not to have a shortcut for an operation, click **Clear**.



## Compiler page

These parameters are detailed in the [Compiler Settings](#) section.

## Database page

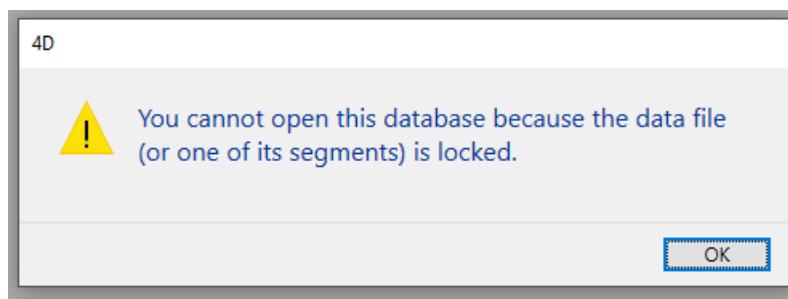
### Data storage page

You use this page to configure data storage on disk for the 4D database.

#### General Settings

##### Allow Read Only Data file Use

This option allows configuration of the application operation when opening a locked data file at the operating system level. 4D includes a mechanism that automatically prevents the opening of a database when its data file, or one of its segments, is locked. In this case, when this detection option is activated, 4D displays a warning message and does not open the database:



Unless this option is checked, it is not possible to open a database when its data file is locked (default operation for 4D databases).

#### About locked files

Locked files can be read but their contents cannot be modified. For example, files are locked when they are stored on a non-rewritable support (DVD type) or when they are copied from this type of support. 4D can work in a transparent manner with locked data files, which allows, more particularly, the use of projects stored on DVD. However, this operation runs the risk of inadvertent use of a locked data file in which modifications will not be saved. This is why by default 4D does not allow databases with a locked data file to be opened.

#### Temporary Folder Location

This area lets you change the location of temporary files created while 4D is running. The temporary files folder is used by the application, when necessary, to temporarily save the data in memory to disk.

The current location of this folder is displayed in the "Current:" area. You can click in this area to show the pathname as a scrolldown list:



Three location options are provided:

- **System:** When this option is selected, the 4D temporary files are created in a

folder placed at the location specified by Windows and/or macOS. You can find out the current location defined by your system using the [Temporary folder](#) 4D command. The files are put into a subfolder whose name consists of the database name and a unique identifier.

- **Data File Folder** (default option): When this option is selected, the 4D temporary files are created in a folder named "temporary files" located at the same level as the data file of the database.
- **User Defined**: This option is used to set a custom location. If the location option is modified, it will be necessary to restart the database in order for the new option to be taken into account. 4D checks whether the folder selected can be write-accessed. If this is not the case, the application tries other options until a valid folder is found.

This option is stored in the "extra properties" of the structure that is available when the structure definition is exported in XML (see [Exporting and importing structure definitions](#)).

## Text comparison

If you change one of these options, you have to quit and reopen the database to make the change effective. Once the database is reopened, all of the database's indexes are automatically re-indexed.

- **Consider @ as a wildcard only when at the beginning or end of text patterns**: Allows you to set how the at sign "@" will be interpreted when used in a query or a character string comparison, when it is located in a word. When this option is not checked (default value), the at sign is used as the wildcard character, in other words, it replaces any character (see [Wildcard character \(@\)](#)).

When the option is checked, the at sign is regarded as a simple character if it is located within a word. This setting is especially useful when searching for E-mail addresses, where the @ sign is used internally. This option has an influence on searches, sorts, string comparisons, as well as on data stored in tables and data found in memory, like arrays. Fields and variables of the alpha (indexed or not) and text type are concerned by how the @ character is interpreted in searches and sorts.

### Notes:

- For searches, if the search criteria begins or ends with @, the "@" character will be treated as a wildcard. Only if the "@" character is placed in the middle of a word (for example: [bill@cgi.com](mailto:bill@cgi.com)) does 4D treat it differently.
  - This option can also have an influence on the behavior of the commands in the [Objects \(Forms\)](#) theme that accept the wildcard character ("@") in the object parameter.
  - For security reasons, only the Administrator or Designer of the database can modify this parameter.
- **Current data language**: Used to configure the language used for character string processing and comparison. The language choice has a direct influence on the sorting and searching of text, as well as the character case, but it has no effect on the translation of texts or on the date, time or currency formats, which remain in the system language. By default, 4D uses the system language.

A 4D project can thus operate in a language different from that of the system. When a project is opened, the 4D engine detects the language used by the data file and provides it to the language (interpreter or compiled mode). Text comparisons, regardless of whether they are carried out by the project engine or

the language, are done in the same language.

You can modify this setting in the application Preferences (see [General Page](#)). In this case, the setting applies to all the new databases created by 4D.

- **Consider only non-alphanumeric chars for keywords:** Modifies the algorithm used by 4D to identify keyword separators and hence build their indexes. By default, when this option is not checked, 4D uses a sophisticated algorithm that takes linguistic characteristics into account.

This algorithm is similar to the one used by word-processing software to determine the boundaries when selecting a word that is double-clicked. For more information about this algorithm, refer to the following address:

<http://userguide.icu-project.org/boundaryanalysis>.

When this option is checked, 4D uses a simplified algorithm. In this configuration, any non-alphanumeric character (i.e., not a letter or a number) is considered as a keyword separator. This setting meets specific requirements associated with certain languages such as Japanese.

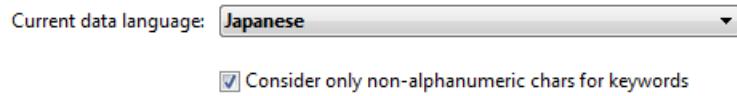
- **Sorting order appropriate for searching:** This option is only displayed when the Japanese language is selected. Modifies the interpretation of characters such as the "Katakana-Hiragana Prolonged Sound Mark" or "長音記号" or the "Japanese Iteration Marks" such as ">" or "々". Typical Japanese speaker is likely to prefer the results when the setting is enabled.

## Support of MeCab (Japanese version)

On Japanese systems, 4D supports the *MeCab* library, with a indexing algorithm for keywords that is particularly suited for the Japanese language.

This algorithm is used by default in Japanese versions of 4D. If needed, you can disable the use of the *MeCab* algorithm and use the conventional *ICU* library.

To disable *MeCab*, just check the **Consider only non-alphanumeric chars for keywords** option:



## Memory page

You use the settings on this tab to configure the cache memory for the database.

### Database Cache Settings

- **Calculation of adaptive cache:** When this option is checked, management of the memory cache is done dynamically by the system --- respecting limits that you set. This allows configuration of a high performance memory cache adapted to most configurations. The size of the memory cache is then calculated dynamically depending on set parameters. The values offered by default correspond to standard 4D usage.
  - **Memory to be reserved for other applications and for the system:** Portion of the RAM memory to reserve for the System and other applications. This value is increased for optimization when other applications are running

on the same machine as 4D.

- **Percentage of available memory used for cache:** Percentage of the remaining memory allocated to the cache by default. To obtain the size allocated by default to the cache, simply perform the following calculation: (Physical memory -- Physical memory to be reserved) X Percentage of the memory used for the cache. In the adaptive mode, the size of the memory cache varies dynamically depending on the needs of the application and the system. You can set limits using the following two options:
  - **Minimum Size:** Minimum amount of memory that must be reserved for the cache. This value cannot be less than 100 MB.
  - **Maximum Size:** Maximum amount of memory that can be used by the cache. This value is virtually unlimited. Setting limits is particularly useful for databases that are distributed on machines for which you do not know the memory configuration a priori. In this case, the limits set let you guarantee a minimum performance in all cases. The following diagram illustrates this behavior:

Example for calculating cache memory: *Physical memory to reserve = 256 MB  
Percentage of the available memory used for the cache = 50%  
Maximum size = 1 GB  
Minimum size = 128 MB*

- **Calculation of adaptive cache not checked:** this mode, you set the size of the memory cache for the database yourself. 4D then displays an entry area that allows setting the memory cache to use as well as information related to the physical memory (RAM available on the machine), the current cache and cache after restart (taking your changes into account).

The size of the memory cache that you enter will be reserved for the 4D database, regardless of the state of machine resources. This setting can be used in certain specific configurations, or when the database is designed to be used on dissimilar systems in terms of memory. In most cases, the adaptive cache offers better performance.

- **Flush Cache every ... Seconds/Minutes:** Specifies the time period between each automatic saving of the data cache, i.e., its writing to disk. 4D saves the data placed in the cache at regular intervals. You can specify any time interval between 1 second and 500 minutes. By default, 4D saves your data every 20 seconds. The application also saves your data to disk each time you change to another environment or exit the application. You can also call the [FLUSH CACHE](#) command to trigger the flush at any moment.

When you anticipate heavy data entry, consider setting a short time interval between saves. In case of a power failure, you will only lose the data entered since the previous save (if the database is running without a log file).

If there is a noticeable slowing down of the database each time the cache is flushed, you need to adjust the frequency. This slowness means that a huge amount of records is being saved. A shorter period between saves would therefore be more efficient since each save would involve fewer records and hence be faster.

By default, 4D displays a small window when the cache is flushed. If you do not want this visual reminder, you can uncheck the **Flushing progress** option on the [Interface page](#).



## **Backup page**

These options are detailed in the [\*\*Backup Settings\*\*](#) chapter.

## Client-server page

The Client-server pages group together parameters related to the use of the database in client-server mode. Naturally, these settings are only taken into account when the database is used in remote mode.

### Network options page

#### Network

##### Publish database at startup

This option lets you indicate whether or not the 4D Server database will appear in the list of published databases.

- When this option is checked (default), the database is made public and appears in the list of published databases (**Available** tab).
- When the option is not checked, the database is not made public and it does not appear in the list of published databases. To connect, users must manually enter the address of the database on the **Custom** tab of the connection dialog box.

If you modify this parameter, you must restart the server database in order for it to be taken into account.

##### Publication name

This option lets you change the publication name of a 4D Server database, *i.e.*, the name displayed on the dynamic **Available** tab of the connection dialog box (see the [Connecting to a 4D Server Database](#) section). By default, 4D Server uses the name of the project file. You can enter any custom name you want.

This parameter is not taken into account in custom client-server applications. In theory, the client application connects directly to the server application, without passing by the connection dialog box. However, in the event of an error, this dialog box can appear; in this case, the publication name of the server application is the name of the compiled project.

##### Port Number

This option lets you change the TCP port number on which 4D Server publishes the database. This information is stored in the project and on each client machine. By default, the TCP port number used by 4D Server and 4D in remote mode is 19813.

Customizing this value is necessary when you want to use several 4D applications on the same machine; in this case, you must specify a different port number for each application. When you modify this value from 4D Server or 4D, it is automatically passed on to all the 4D machines connected to the database.

To update any other client machines that are not connected, you just need to enter the new port number (preceded by a colon) after the IP address of the server machine on the **Custom** tab of the connection dialog box at the time of the next connection. For example, if the new port number is 19888:

Only databases published on the same port as the one set in 4D client are visible on the TCP/IP dynamic publication page.

## 4D Server and port numbers

4D Server uses three TCP ports for communications between internal servers and clients:

- **SQL Server:** 19812 by default (can be modified via the "SQL/Configuration" page of the Preferences).
- **Application Server:** 19813 by default (can be modified via the "Client-Server/Configuration" page of the Preferences, see above).
- **DB4D Server** (database server): 19814 by default . This port number cannot be modified directly but it always consists of the application server port number + 1. When a 4D client connects to 4D Server, it uses the TCP port of the application server (19813 or the port indicated after the colon ':' in the IP address shown in the connection dialog box). Connection to other servers via their respective ports is then automatic; it is no longer necessary to specify them. Note that in the case of access via a router or a firewall, the three TCP ports must be opened explicitly.

## Authentication of user with domain server

This option allows you to implement SSO (*Single Sign On*) capabilities in your 4D Server database on Windows. When you check this option, 4D transparently connects to the Active directory of the Windows domain server and gets the available authentication tokens. This option is described in the [Single Sign On \(SSO\) on Windows](#) section.

## Service Principal Name

When Single Sign On (SSO) is enabled (see above), you must fill in this field if you want to use Kerberos as your authentication protocol. This option is described in the [Single Sign On \(SSO\) on Windows](#) section.

## Client-Server Connections Timeout

This device is used to set the timeout (period of inactivity beyond which the connection is closed) between 4D Server and the client machines connecting to it. The Unlimited option removes the timeout. When this option is selected, client activity control is eliminated.

When a timeout is selected, the server will close the connection of a client if it does not receive any requests from the latter during the specified time limit.

## Client-Server Communication

### Register Clients at Startup For Execute On Client

When this option is checked, all the 4D remote machines connecting to the database can execute methods remotely. This mechanism is detailed in the section [Stored procedures on client machines](#).

### Encrypt Client-Server Communications

This option lets you activate the secured mode for communications between the server machine and the 4D remote machines. This option is detailed in the [Encrypting](#)

[Client/Server Connections](#) section.

## Update Resources folder during a session

This setting can be used to globally set the updating mode for the local instance of the **Resources** folder on the connected 4D machines when the **Resources** folder of the database is modified during the session (the **Resources** folder is automatically synchronized on the remote machine each time a session is opened). Three settings are available:

- **Never:** The local **Resources** folder is not updated during the session. The notification sent by the server is ignored. The local **Resources** folder may be updated manually using the **Update Local Resources** action menu command (see [Using the Resources explorer](#)).
- **Always:** The synchronization of the local **Resources** folder is automatically carried out during the session whenever notification is sent by the server.
- **Ask:** When the notification is sent by the server, a dialog box is displayed on the client machines, indicating the modification. The user can then accept or refuse the synchronization of the local **Resources** folder. The **Resources** folder centralizes the custom files required for the database interface (translation files, pictures, etc.). Automatic or manual mechanisms can be used to notify each client when the contents of this folder have been modified. For more information, please refer to the [Managing the Resources folder](#) section.

## IP configuration page

### Allow-Deny Configuration Table

This table allows you to set access control rules for the database depending on 4D remote machine IP addresses. This option allows reinforcing security, for example, for strategic applications.

This configuration table does not control Web connections.

The behavior of the configuration table is as follows:

- The "Allow-Deny" column allows selecting the type of rule to apply (Allow or Deny) using a pop-up menu. To add a rule, click on the Add button. A new row appears in the table. The **Delete** button lets you remove the current row.
- The "IP Address" column allows setting the IP address(es) concerned by the rule. To specify an address, click in the column and enter the address in the following form: 123.45.67.89 (IPv4 format) or 2001:0DB8:0000:85A3:0000:AC1F:8001 (IPv6 format). You can use an *(asterisk) character to specify "starts with" type addresses. For example, 192.168.* indicates all addresses starting with 192.168.
- The application of rules is based on the display order of the table. If two rules are contradictory, priority is given to the rule located highest in the table. You can re-order rows by modifying the current sort (click the header of the column to alternate the direction of the sort). You can also move rows using drag and drop.
- For security reasons, only addresses that actually match a rule will be allowed to connect. In other words, if the table only contains one or more Deny rules, all addresses will be refused because none will match at least one rule. If you want to deny only certain addresses (and allow others), add an Allow \* rule at the end of the table. For example:
  - Deny 192.168.\* (deny all addresses beginning with 192.168)
  - Allow \* (but allow all other addresses)

By default, no connection restrictions are applied by 4D Server: the first row of the

table contains the Allow label and the \* (all addresses) character.

## Web page

Using the tabs on the **Web** page, you can configure various aspects of the integrated Web server of 4D (security, startup, connections, Web services, etc.). For more information about how the 4D Web server works, see [Web server](#). For more information about 4D Web services, refer to the [Publication and use of Web Services](#) chapter.

## Configuration

### Publishing Information

#### Launch Web Server at Startup

Indicates whether the Web server will be launched on startup of the 4D application. This option is described in the [Web server administration](#) section.

#### Enable HTTP

Indicates whether or not the Web server will accept non-secure connections. See [Enable HTTP](#).

#### HTTP Port

Listening IP (TCP) port number for HTTP. See [HTTP Port](#).

#### IP Address

IP address on which the 4D web server will receive HTTP requests (4D local and 4D Server). See [IP Address to listen](#).

#### Enable HTTPS

Indicates whether or not the Web server will accept secure connections. See [Enable HTTPS](#).

#### HTTPS Port

Allows you to modify the TCP/IP port number used by the Web server for secured HTTP connections over TLS (HTTPS protocol). See [HTTPS Port](#).

#### Allow database access through 4DSYNC URLs

*Compatibility Note:* This option is [deprecated](#). For database access through HTTP, it is now recommended to use ORDA remote datastore features and REST requests.

#### Paths

##### Default HTML Root

Define the default location of the Web site files and to indicate the hierarchical level on the disk above which the files will not be accessible. See [Root Folder](#).

##### Default Home Page

Designate a default home page for the Web server. See [Default Home page](#).

## Options (I)

### Cache

#### Use the 4D Web cache

Enables the web page cache. See [Cache](#).

#### Pages Cache Size

Sets the cache size. See [Cache](#).

#### Clear Cache

At any moment, you can clear the cache of the pages and images that it contains (if, for example, you have modified a static page and you want to reload it in the cache). To do so, you just have to click on the **Clear Cache** button. The cache is then immediately cleared.

You can also use the special URL </4DCACHECLEAR>.

### Web Process

This area allows you to configure how the web server will handle user sessions and their associated processes.

The **Legacy sessions** option is only available for compatibility in databases/projects created with 4D versions prior to 4D v18 R6.

#### Scalable sessions (multi-process sessions)

When you select this option (recommended), a user session is managed through a **Session** object. See the [User sessions page](#).

#### No sessions

When this option is selected, the web server does not provide any specific support for [user sessions](#). Successive requests from web clients are always independent and no context is maintained on the server.

In this mode, you can configure additional web server settings:

- [Maximum Concurrent Web Processes](#)
- [Reuse Temporary Contexts \(4D in remote mode\)](#)
- [Use preemptive processes](#)

#### Legacy sessions (single process sessions)

*Compatibility Note:* This option is only available in databases/projects created with a 4D version prior to 4D v18 R6.

This option enables the handling of legacy user sessions by the 4D HTTP server. This mechanism is described in the [Web Sessions Management \(Legacy\)](#) section. See [Keep Session](#).

When selected, the [Reuse Temporary Contexts \(4D in remote mode\)](#) option is

automatically checked (and locked).

## Maximum Concurrent Web Processes

Not available with [scalable sessions](#).

Strictly high limit of concurrent web processes. See [Maximum Concurrent Web Processes](#).

## Reuse Temporary Contexts

Not available with [scalable sessions](#).

Allows you to optimize the operation of the 4D Web server in remote mode. See [Reuse temporary contexts in remote mode](#)).

## Use preemptive processes

Not available with [scalable sessions](#).

Enables preemptive web processes in your compiled applications. When **Use preemptive processes** is selected, the eligibility of your web-related code (including 4D tags and web database methods) to the preemptive execution will be evaluated during the compilation. For more information, see [Using preemptive Web processes](#).

This option does not apply to scalable sessions, REST processes (compiled mode), and web service processes (server or client). See [Enabling the preemptive mode for the web server](#).

## Inactive Process Timeout

Not available with [scalable sessions](#).

Allows you to set the maximum timeout before closing for inactive Web processes on the server. See [Inactive Process Timeout](#).

## Web Passwords

Set the authentication system that you want to apply to your Web server. Three options are proposed:

Custom (default) Passwords with BASIC protocol Passwords with DIGEST protocol

Using **Custom** authentication is recommended. See [Authentication](#) chapter in the *Web Development* documentation.

## Options (II)

### Text Conversion

#### Send Extended Characters Directly

See [Deprecated Settings](#).

#### Standard Set

Define the set of characters to be used by the 4D Web server. See [Character Set](#).

## Keep-Alive Connections

See [Deprecated Settings](#).

## CORS Settings

### Enable CORS

Enables Cross-origin resource sharing (CORS) service. See [Enable CORS Service](#).

### Domain names/HTTP methods allowed

List of allowed hosts and methods for the CORS service. See [CORS Settings](#).

## Log (type)

### Log Format

Starts or stops the recording of requests received by the 4D web server in the *logweb.txt* file and sets its format. See [Log Recording](#).

The activation and deactivation of the log file of requests can also be carried out by programming using the [WEB SET OPTION](#) command.

The log format menu provides the following options:

- **No Log File:** When this option is selected, 4D will not generate a log file of requests.
- **CLF (Common Log Format):** When this option is selected, the log of requests is generated in CLF format. With the CLF format, each line of the file represents a request, such as:  
host rfc931 user [DD/MMM/YYYY:HH:MM:SS] "request" state length  
Each field is separated by a space and each line ends by the CR/LF sequence (character 13, character 10).
  - host: IP address of the client (ex. 192.100.100.10)
  - rfc931: information not generated by 4D, it's always - (a minus sign)
  - user: user name as it is authenticated, or else it is - (a minus sign). If the user name contains spaces, they will be replaced by \_ (an underscore).
  - DD: day, MMM: a 3-letter abbreviation for the month name (Jan, Feb,...), YYYY: year, HH: hour, MM: minutes, SS: seconds

The date and time are local to the server.

- request: request sent by the client (ex. GET /index.htm HTTP/1.0)
- state: reply given by the server.
- length: size of the data returned (except the HTTP header) or 0.

**Note:** For performance reasons, the operations are saved in a memory buffer in packets of 1Kb before being written to disk. The operations are also written to disk if no request has been sent every 5 seconds. The possible values of state are as follows: 200: OK 204: No contents 302: Redirection 304: Not modified 400: Incorrect request 401: Authentication required 404: Not found 500: Internal error The CLF format cannot be customized.

- **DLF (Combined Log Format):** When this option is selected, the request log is generated in DLF format. DLF format is similar to CLF format and uses exactly the same structure. It simply adds two additional HTTP fields at the end of each request: Referer and User-agent.

- Referer: Contains the URL of the page pointing to the requested document.
- User-agent: Contains the name and version of the browser or software of the client at the origin of the request.

The DLF format cannot be customized.

- **ELF (Extended Log Format):** When this option is selected, the request log is generated in ELF format. The ELF format is very widespread in the world of HTTP browsers. It can be used to build sophisticated logs that meet specific needs. For this reason, the ELF format can be customized: it is possible to choose the fields to be recorded as well as their order of insertion into the file.
- **WLF (WebStar Log Format):** When this option is selected, the request log is generated in WLF format. WLF format was developed specifically for the 4D WebSTAR server. It is similar to the ELF format, with only a few additional fields. Like the ELF format, it can be customized.

**Configuring the fields** When you choose the ELF (Extended Log Format) or WLF (WebStar Log Format) format, the "Weg Log Token Selection" area displays the fields available for the chosen format. You will need to select each field to be included in the log. To do so, use the arrow buttons or simply drag and drop the desired fields into the "Selected Tokens" area.

**Note:** You cannot select the same field twice.

The following table lists the fields available for each format (in alphabetical order) and describes its contents:

Field	ELF	WLF	Value
BYTES_RECEIVED	X		Number of bytes received by the server
BYTES_SENT	X	X	Number of bytes sent by the server to the client
C_DNS	X	X	IP address of the DNS (ELF: field identical to the C_IP field)
C_IP	X	X	IP address of the client (for example 192.100.100.10)
CONNECTION_ID		X	Connection ID number
CS(COOKIE)	X	X	Information about cookies contained in the HTTP request
CS(HOST)	X	X	Host field of the HTTP request
CS(REFERER)	X	X	URL of the page pointing to the requested document
CS(USER_AGENT)	X	X	Information about the software and operating system of the client
CS_SIP	X	X	IP address of the server
CS_URI	X	X	URI on which the request is made
CS_URI_QUERY	X	X	Request query parameters
CS_URI_STEM	X	X	Part of request without query parameters
DATE	X	X	DD: day, MMM: 3-letter abbreviation for month (Jan, Feb, etc.), YYYY: year
METHOD	X	X	HTTP method used for the request sent to the server
PATH_ARGS		X	CGI parameters: string located after the "\$" character
STATUS	X	X	Reply provided by the server
TIME	X	X	HH: hour, MM: minutes, SS: seconds
TRANSFER_TIME	X	X	Time requested by server to generate the reply
USER	X	X	User name if authenticated; otherwise - (minus sign). If the user name contains spaces, they are replaced by _ (underlines)
URL		X	URL requested by the client

Dates and times are given in GMT.

## Log (backup)

Configure the automatic backup parameters for the request log. First you must choose the frequency (days, weeks, etc.) or the file size limit criterion by clicking on the corresponding radio button. You must then specify the precise moment of the backup if necessary.

- **No Backup:** The scheduled backup function is deactivated.
- **Every X hour(s):** This option is used to program backups on an hourly basis. You can enter a value between 1 and 24 .
  - **starting at:** Used to set the time at which the first back up will begin.
- **Every X day(s) at X:** This option is used to program backups on a daily basis. Enter 1 if you want to perform a daily backup. When this option is checked, you must indicate the time when the backup must be started.
- **Every X week(s), day at X:** This option is used to program backups on a weekly basis. Enter 1 if you want to perform a weekly backup. When this option is checked, you must indicate the day(s) of the week and the time when each backup must be started. You can select several days of the week if desired. For example, you can use this option to set two weekly backups: one on Wednesdays and one on Fridays.
- **Every X month(s), Xth day at X:** This option is used to program backups on a monthly basis. Enter 1 if you want to perform a monthly backup. When this option is checked, you must indicate the day of the month and the time when the backup must be started.
- **Every X MB:** This option is used to program backups based on the size of the current request log file. A backup is automatically triggered when the file reaches the set size. You can set a size limit of 1, 10, 100 or 1000 MB.

In the case of scheduled backups, if the Web server was not launched when the backup was scheduled to occur, on the next startup 4D considers the backup as failed and applies the appropriate settings, set via the Database Settings.

## Web Services

You use the options on this tab to activate and configure Web services for the 4D project, both for their publishing (server side) and their subscription (client side).

For more information about the support of Web Services in 4D, refer to the [Publication and use of Web Services](#) chapter.

### Server Side

This area contains various options related to the use of 4D as a Web Services "server" i.e., publishing project methods in the form of Web Services.

- **Allow Web Services Requests:** This option lets you initialize the publication of Web Services. If this option has not been checked, 4D refuses SOAP requests and does not generate a WSDL - even if methods have the *Published in WSDL* attribute. When this option is checked, 4D creates the WSDL file.
- **Web Service Name:** This area lets you change the "generic name" of the Web

Service. This name is used to differentiate the services both at the SOAP server level (when the server publishes several different Web Services), as well as in the Web Services directories. By default, 4D uses the name A\_WebService.

- **Web Services Namespace:** This area is used to change the namespace of the Web Services published by 4D. Each Web Service published on the Internet must be unique. The uniqueness of the names of Web Services is ensured by using XML namespaces. A namespace is an arbitrary character string used to identify a set of XML tags in a unique way. Typically, the namespace begins with the URL of the company (<http://mycompany.com/mynamespace>). In this case, it is not indispensable to have anything in particular at the URL indicated; what matters is that the character string used is unique. By default, 4D uses the following namespace: <http://www.4d.com/namespace/default>.

In conformity with the XML standard for tag names, the character strings used must not contain spaces nor start with a number. Moreover, to avoid any risk of incompatibility, we recommend that you do not use any extended characters (such as accented characters).

## Client Side

This area contains various options related to the use of 4D as a Web Services "client" i.e., subscribing to services published on the network.

- **Wizard Method Prefix:** This area lets you change the prefix that is added automatically by 4D to the name of proxy methods generated by the Web Services Wizard. Proxy project methods form a link between the 4D application and the Web Services server. By default, 4D uses the prefix "proxy\_".

## Web Features

This page contains the options used to enable and control advanced Web features such as the REST server.

### Publishing

#### Expose as REST server

Starts and stops the REST Server. See [REST Server Configuration](#).

### Access

This option specifies a group of 4D users that is authorized to establish the link to the 4D database using REST requests. See [Configuring REST access](#).

### Web Studio

#### Enable access to the web studio

Enables general access to the web studio. You still need to configure it at every project level.

## **SQL page**

This page is used to configure the publishing parameters, access rights, and engine options of the [4D SQL Server](#).

### **SQL Server Publishing**

See the [Configuration of 4D SQL Server](#) page on doc.4d.com.

### **SQL Access Control for the default schema**

See the [Configuration of 4D SQL Server](#) page on doc.4d.com.

### **SQL Engine Options**

See the [SQL Engine Options](#) paragraph on doc.4d.com.

## PHP page

In 4D, you can execute PHP scripts directly by configuring the PHP page of the Database Settings (see [Executing PHP scripts in 4D](#) in the 4D *Language Reference* manual).

## Interpreter

- **IP Address** and Port number By default, 4D provides a PHP interpreter, compiled in FastCGI. For reasons related to the internal architecture, execution requests go to the PHP interpreter at a specific HTTP address. By default, 4D uses the address 127.0.0.1 and port 8002. You can change this address and/or port if they are already used by another service or if you have several interpreters on the same machine. To do this, you modify the **IP Address** and **Port number** parameters. Note that the HTTP address must be on the same machine as 4D.
- **External interpreter** If you use an external PHP interpreter, it must be compiled in FastCGI and be on the same machine as 4D (see "Using another PHP interpreter or another php.ini file" in [Executing PHP scripts in 4D](#)). Select this option so 4D does not attempt a connection with the internal interpreter when executing a PHP request. Note that this configuration requires your manual execution and control of the external interpreter.

**4D Server:** These settings are shared between 4D Server and the 4D remote machines so it is not possible to use an external interpreter on the server machine and simultaneously use the internal interpreter on the client machines (and vice versa). Also, if the server uses an external interpreter on port 9002, the client machines must also use an interpreter on this port.

## Options

These options are related to the automatic management of the 4D PHP interpreter and are disabled when the **External interpreter** option is selected.

- **Number of processes:** The 4D PHP interpreter drives a set of system execution processes called "child processes". For optimization, it can run and keep up to five child processes simultaneously by default. You can modify the number of child processes according to your needs. For example, you may want to increase this value if you call on the PHP interpreter intensively. For more information, refer to the "Architecture" section in [Executing PHP scripts in 4D](#).

**Note:** Under Mac OS, all child processes share the same port. Under Windows, each child process uses a specific port number. The first number is the one set for the PHP interpreter; the other child processes increment this number. For example, if the default port is 8002 and you launch 5 child processes, they will use ports 8002 to 8006.

- **Restart the interpreter after X requests:** This sets the maximum number of requests that the 4D PHP interpreter accepts. When this number is reached, the interpreter restarts. For more information about this parameter, refer to the FastCGI-PHP documentation.

**Note:** In this dialog box, the parameters are specified by default for all connected machines and all sessions. You can also modify and read them separately for each machine and each session using the [SET](#)

[DATABASE PARAMETER](#) and [Get database parameter](#) commands. The parameters modified by the [SET DATABASE PARAMETER](#) command have priority for the current session.

## Security page

This page contains options related to data access and protection for your desktop applications.

**Note:** For a general overview of 4D's security features, see the [4D Security guide](#).

## Data Access / Remote Users Access

These settings do not apply to project databases opened in single-user mode.

- **Design and Runtime Explorer Access:** Gives the specified group the ability to enter the Design environment of the database and display the Runtime Explorer.

Note that:

- Setting an access group in the Design environment also lets you deactivate the **Create table** option in the data import dialog box. For more information about this dialog box, refer to [Importing data from files](#).
- The Designer and Administrator always have access to the Design environment and Runtime Explorer, even if they are not explicitly part of the specified access group. For more information about users and user groups, refer to the [Users and groups](#) chapter.
- **Default User:** When a Default User has been set, every user that opens the database or logs onto it has the same access privileges and restrictions defined for this Default User. It is no longer necessary to enter a user name. Moreover, if you have not associated a password with the Default User, the Password dialog box no longer appears and the database opens directly. This option simplifies access to the database while maintaining a complete data control system.
  - If you have associated a password with the Default User, a dialog box appears when the database is opened and the users must enter a password.
  - If you haven't associated a password with the Default User, the User Identification dialog box will not appear. **Note:** You can "force" the display of the User Identification dialog box when the "Default User" mode is active, for instance in order to connect as Administrator or Designer. To do so, press the **Shift** key while opening the database or connecting to it.
- **Display User List in Password Dialog Box:** If this option is checked, users must choose their name from the list of users and enter their password in the User Identification dialog box. If it is not checked, users must enter both their name and password. For more information about the two versions of the password dialog box, see the section "Access system overview" in [Access system overview](#).
  - **User List in Alphabetical Order** (only available if the previous option is checked): When this option is checked, the list of users in the password entry dialog box is sorted by alphabetical order.
- **Users can change their password:** When this option is checked, a **Change** button is displayed in the User Identification dialog box. This button lets the user access a dialog box that can be used to change their password (for more information about this dialog box, refer to the "Modification of password by user" in [Ensuring system maintenance](#)). If desired, you can hide the **Change** button so

that users cannot modify their passwords. To do so, just uncheck this option.

## Options

- **Filtering of commands and project methods in the formula editor and 4D Write Pro documents:** For security reasons, by default 4D restricts access to the commands, functions and project methods in the [Formula editor](#) in Application mode or added to multistyle areas or 4D Write Pro documents using the [ST INSERT EXPRESSION](#) command: only certain 4D functions and project methods that have been explicitly declared using the [SET ALLOWED METHODS](#) command can be used. You can completely or partially remove this filtering using the following options.
  - **Enabled for all** (default option): Access to commands, functions and project methods is restricted for all users, including the Designer and the Administrator.
  - **Disable for the Designer and the Administrator:** This option grants full access to 4D commands and to methods only for the Designer and Administrator. It can be used to set up an unlimited access mode to commands and methods while remaining in control of the operations carried out. During the development phase, this mode can be used to freely test all the formulas, reports, and so on. During operation, it can be used to set up secure solutions that allow access to commands and methods on a temporary basis. This consists in changing the user (via the [CHANGE CURRENT USER](#) command) before calling a dialog box or starting a printing process that requires full access to the commands, then returning to the original user when the specific operation is completed. **Note:** If full access has been enabled using the previous option, this option will have no effect.
  - **Disabled for all:** This option disables control within formulas. When this option is checked, users have access to all the 4D commands and plug-ins as well as all project methods (except for invisible ones). **Note:** This option takes priority over the [SET ALLOWED METHODS](#) command. When it is checked, this command does nothing.
- **Enable User Settings:** You need to check this option to be able to display separated dialog boxes for user settings. When this option is checked, up to three dialog boxes are available: **Structure Settings**, **User Settings**, and **User Settings for Data File**. For more information, refer to [User settings](#).
- **Execute "On Host Database Event" method of the components:** The [On Host Database Event database method](#) facilitates the initialization and backup phases for 4D components. For security reasons, you must explicitly authorize the execution of this method in each host database. To do this, you must check this option. By default, it is not checked.

When this option is checked:

- 4D components are loaded,
- each [On Host Database Event database method](#) of the component (if any) is called by the host database,
- the code of the method is executed.

When it is not checked:

- 4D components are loaded but they have to manage their initialization and backup phases themselves.
- the developer of the component has to publish the component methods that must be called by the host database during these phases (startup and

shutdown)

- the developer of the host database must call the appropriate methods of the component at the right time (must be covered in the component documentation).

## Compatibility page

The Compatibility page groups together parameters related to maintaining compatibility with previous versions of 4D.

The number of options displayed depends on the version of 4D with which the original database/project was created, as well as the settings modified in this database/project.

This page lists the compatibility options available for database/projects converted from 4D v18 onwards. For older compatibility options, refer to the [Compatibility page](#) on [doc.4d.com](#).

- **Use legacy network layer:** Starting with 4D v15, 4D applications propose a new network layer, named *ServerNet*, to handle communications between 4D Server and remote 4D machines (clients). The former network layer has become obsolete, but it is kept to ensure compatibility with existing databases. Using this option, you can enable the former network layer at any time in your 4D Server applications depending on your needs. *ServerNet* is used automatically for new databases and databases converted from a v15 release or later. Note that in case of a modification, you need to restart the application for the change to be taken into account. Any client applications that were logged must also be restarted to be able to connect with the new network layer. **Note:** This option can also be managed by programming using the `SET DATABASE PARAMETER` command.
- **Use standard XPath:** By default this option is unchecked for databases converted from a 4D version prior to v18 R3, and checked for databases created with 4D v18 R3 and higher. Starting with v18 R3, the XPath implementation in 4D has been modified to be more compliant and to support more predicates. As a consequence, non-standard features of the previous implementation no longer work. They include:
  - initial "/" is not the root node only - using a / as first character in a XPath expression does not declare an absolute path from the root node
  - no implicit current node - the current node has to be included in the XPath expression
  - no recursive searches in repeated structures - only the first element is parsed.¥

Although not standard, you might want to keep using these features so that your code continues to work as before -- in this case, just set the option *unchecked*. On the other hand, if your code does not rely on the non-standard implementation and if you want to benefit from the extended XPath features in your databases (as described in the [DOM Find XML element](#) command), make sure the **Use standard XPath** option is *checked*.

- **Use LF for end of line on macOS:** Starting with 4D v19 R2 (and 4D v19 R3 for XML files), 4D writes text files with line feed (LF) as default end of line (EOL) character instead of CR (CRLF for xml SAX) on macOS in new projects. If you want to benefit from this new behavior on projects converted from previous 4D versions, check this option. See [TEXT TO DOCUMENT](#), [Document to text](#), and [XML SET OPTIONS](#).
- **Don't add a BOM when writing a unicode text file by default:** Starting with 4D v19 R2 (and 4D v19 R3 for XML files), 4D writes text files without a byte order mark (BOM) by default. In previous versions, text files were written with a BOM

by default. Select this option if you want to enable the new behavior in converted projects. See [TEXT TO DOCUMENT](#), [Document to text](#), and [XML SET OPTIONS](#).

- **Map NULL values to blank values unchecked by default a field creation:** For better compliance with ORDA specifications, in databases created with 4D v19 R4 and higher the **Map NULL values to blank values** field property is unchecked by default when you create fields. You can apply this default behavior to your converted databases by checking this option (working with Null values is recommended since they are fully supported by [ORDA](#)).

## Preferences

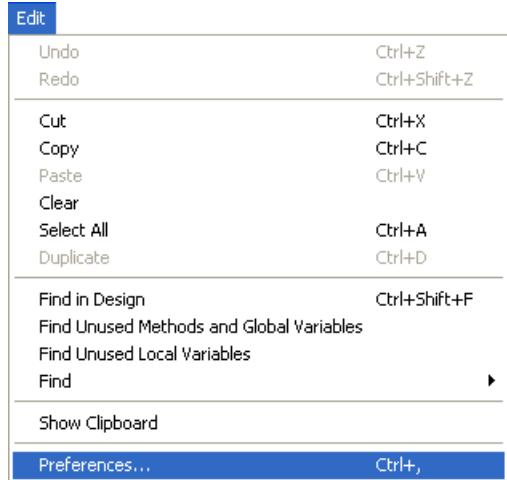
User preferences specify various settings affecting your working environment, e.g. default options, display themes, Code Editor features, shortcuts, etc. They are applied to all projects opened with your 4D or 4D Server application.

**4D Server:** Object locking occurs when two or more users try to modify the settings in the Preferences dialog box at the same time. Only one user can use the Preferences dialog box at a time.

4D offers a different set of parameters specific to the open project: **Settings** (available from the **Design** menu). For more information, refer to the [Settings chapter](#).

## Access

You can access the Preferences dialog box from the **Edit > Preferences...** menu (Windows) or the **4D** Application menu (macOS):



This menu option is available even when there is no open project.

You can also display the Preferences dialog box in Application mode using the "Preferences" standard action (associated with a menu item or a button) or using the `OPEN SETTINGS WINDOW` command.

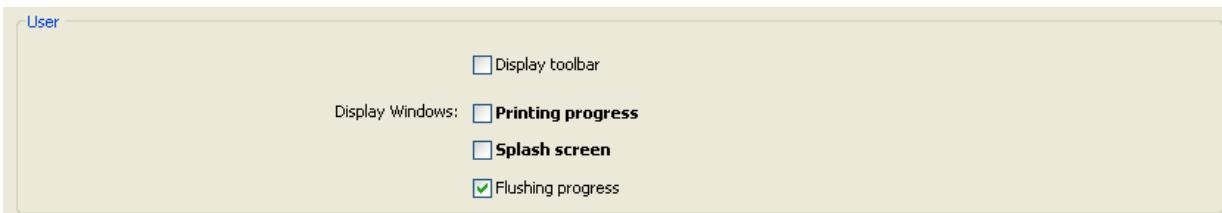
## Storage

Settings made in the Preferences dialog box are saved in an XML format preferences file named **4D Preferences vX.X.4DPreferences** that is stored in the active 4D folder of the current user, as returned by the [Get 4D folder](#) command:

- Windows: `{disk}\Users\{UserName}\AppData\Roaming\4D`
- macOS: `{disk}:Users:{UserName}:Library:Application Support:4D`

## Customizing parameters and reset settings

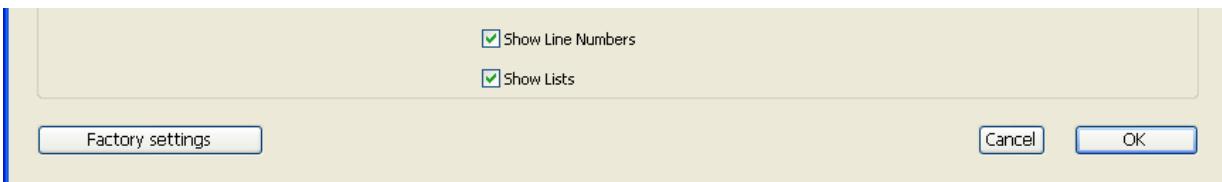
In settings dialog boxes, parameters whose values have been modified appear in bold:



Preferences indicated as customized may have been modified directly in the dialog box, or may have been modified previously in the case of a converted database.

A parameter still appears in bold even when its value is replaced manually with its default values. This way it is always possible to visually identify any parameters that have been customized.

To reset the parameters to their default values and remove the bold style indicating that they have been customized, click on the **Reset to factory settings** button:



This button resets all the parameters of the current page. It becomes active when at least one parameter has been modified on the current page.

## General Page

This page contains various options to configure the general operation of your 4D application.

## Options

### At startup

This option allows you to configure the default 4D display at startup, when the user launches only the application.

- **Do nothing:** Only the application window appears, empty.
- **Open Local Project dialog:** 4D displays a standard open document dialog box, allowing you to select a local project.
- **Open last used project:** 4D directly opens the last project used; no opening dialog box appears.

To force the display of the opening dialog box when this option is selected, hold down the **Alt** (Windows) or **Option** (macOS) key while launching the project.

- **Open Remote Project dialog:** 4D displays the standard 4D Server logon dialog, allowing you to select a project published on the network.
- **Open Welcome Wizard dialog** (factory setting): 4D displays the Welcome Wizard dialog box.

**4D Server:** The 4D Server application ignores this option. In this environment, the **Do nothing** mode is always used.

### Automatic form creation

This option is only used in binary databases; it is ignored in project architecture. See doc.4d.com.

### Window tabbing (macOS only)

Starting with macOS Sierra, Mac applications can benefit from the Automatic Window Tabbing feature that helps organizing multiple windows: document windows are stacked into a single parent window and can be browsed through tabs. This feature is useful on small screens and/or when using a trackpad.

You can benefit from this feature in the following environments (with 4D 64-bit versions only):

- Code Editor windows
- Form Editor windows

All windows from these editors can be put in tab form:

A set of commands in the **Window** menu allows managing the tabs:

In the 4D's Preferences dialog box, the **Window tabbing** option allows you to control

this feature:

Three options are available:

- **According to System Preferences** (default): 4D windows will behave like defined in the macOS System Preferences (In full screen, Always, or Manually).
- **Never**: Opening a new document in 4D form editor or Code Editor will always result in creating a new window (tabs are never created).
- **Always**: Opening a new document in 4D form editor or method editors will always result in creating a new tab.

### **Appearance (macOS only)**

This menu lets you select the color scheme to use for the **4D development** environment. The specified scheme will be applied to all editors and windows of the Design mode.

You can also set the color scheme to use in your **desktop applications** in the "Interface" page of the Settings dialog box.

Three options are available:

- **According to System Color Scheme Preferences** (default): Use the color scheme defined in the macOS System Preferences.
- **Light**: Use the Light Theme
- **Dark**: Use the Dark Theme

This preference is only supported on macOS. On Windows, the "Light" scheme is always used.

### **Exit Design when going to Application Environment**

If this option is checked, when the user switches to the Application environment using the **Test Application** menu command, all the windows of the Design environment are closed. If this option is not checked (factory setting), the windows of the Design environment remain visible in the background of the Application environment.

### **Enable binary database creation**

If you check this option, two items are added in the **File > New** menu and the **New** toolbar button:

- **Database...**
- **Database from Structure Definition...**

These items allow you to create binary databases (see [Creating a new database](#) section). They are no longer proposed by default because 4D recommends using project-based architecture for new developments.

### **When creating a new project**

#### **Use Log File**

When this option is checked, a log file is automatically started and used when a new database is created. For more information, please refer to [Log file \(.journal\)](#).

## Create package

When this option is checked, 4D databases are automatically created in a folder suffixed .4dbase.

Thanks to this principle, under macOS the database folders appear as packages having specific properties. Under Windows, this has no particular impact.

## Include tokens in project source files

When this option is checked, saved [method source files](#) in new 4D projects will contain **tokens** for classic language and database objects (constants, commands, tables and fields). Tokens are additional characters such as `:C10` or `:5` inserted in the source code files, that allow renaming tables and fields and identifying elements whatever the 4D version (see [Using tokens in formulas](#)).

If you intend to use VCS or external code editors with your new projects, you might want to uncheck this option for a better readability of the code with these tools.

This option can only be applied to projects (binary databases always include tokens).

You can always get the code with tokens by calling [METHOD\\_GET\\_CODE](#) with 1 in the *option* parameter.

## Excluding tokens in existing projects

You can configure your existing projects to save code **without tokens** by inserting the following key in the [`<applicationName>.4DProject`](#) file using a text editor:

```
"tokenizedText": false
```

This setting is only taken into account when methods are saved. Existing methods in your projects are left untouched, unless you resave them.

## Create `.gitignore` file

You might need or want git to ignore some files in your new projects.

You can set this preference by checking the **Create .gitignore file** option.

When a project is created in 4D and that box is checked, 4D creates a `.gitignore` file at the same level as the [`Project`](#) folder (see [Architecture of a Project](#)).

You can define the default contents of the `.gitignore` file by clicking the pencil icon. This will open the `.gitignore` configuration file in your text editor. The contents of this file will be used to generate the `.gitignore` files in your new projects.

The [official git documentation](#) is a great resource to understand how `.gitignore` files work.

## Language of text comparison

This parameter configures the default language used for character string processing and comparison in new databases. The language choice has a direct influence on the sorting and searching of text, as well as the character case, but it has no effect on the translation of texts or on the date, time or currency formats, which remain in the system language. By default (factory setting), 4D uses the current user language set in the system.

A 4D database can thus operate in a language different from that of the system. When a database is opened, the 4D engine detects the language used by the data file and provides it to the language (interpreter or compiled mode). Text comparisons, regardless of whether they are carried out by the database engine or the language, are done in the same language.

When creating a new data file, 4D uses the language previously set in this menu. When opening a data file that is not in the same language as the structure, the data file language is used and the language code is copied into the structure.

You can modify this parameter for the open database using the Database Settings (see [Text comparison](#)).

## Documentation Location

This area configures access to the 4D HTML documentation displayed in your current browser:

- When you hit the **F1** key while the cursor is inserted in a 4D class function or command name in the Code Editor;
- When you double-click on a 4D command in the **Commands Page** of the Explorer.

### Documentation language

Language of the HTML documentation to display. You can select a documentation in a different language from the application language.

#### Look in the local folder first

This option is only taken into account for command documentation access (excluding class functions).

Sets where 4D will look for documentation pages.

- When checked (default), 4D first looks for the page in the local folder (see below). If it is found, 4D displays the page in the current browser. If not, 4D automatically looks for it in the on-line documentation Web site. This makes it possible to access the documentation even when you are offline.
- When not checked, 4D looks for the desired page directly in the on-line documentation Web site and displays it in the current browser. If it is not found, 4D displays an error message in the browser.

#### Local folder

This option is only taken into account for command documentation access (excluding class functions).

Indicates the location of the static HTML documentation. By default, this is the `¥Help¥Command¥language` subfolder. You can view the location by clicking on the menu associated with the area. If this subfolder is not present, the location is shown in red.

You can modify this location as desired, for example if you want to display the documentation in a language different from that of the application. The static HTML documentation can be located on another volume, on a web server, etc. To designate a different location, click on the [...] button next to the entry area and choose a documentation root folder (folder corresponding to the language: `fr`, `en`, `es`, `de` or `ja`).



## Structure Page

### Primary key

These options in the preferences modify the default name and type of the primary key fields that are added automatically by 4D when new tables are created or by means of the [Primary key manager](#).

The following options are available:

- **Name** ("ID" by default): Sets the default name of primary key fields. You can use any name you want, as long as it respects the [4D naming rules](#).
- **Type** ([Longint](#) by default): Sets the default type of primary key fields. You can choose the UUID type. In this case, the primary key fields created by default are of the [Alpha type](#) and have the **UUID Format** and **Auto UUID** field properties checked.

## Structure editor

This group of options configures the display of the 4D Structure editor.

### Graphic quality of the structure

This option varies the level of graphic detail in the Structure editor. By default, the quality is set to **High**. You can select Standard quality in order to give priority to display speed. The effect of this setting is mainly perceptible when using the zoom function (see the "Zoom" paragraph in [Structure editor](#)).

### When a folder is dimmed, its contents are:

This option sets the appearance of dimmed tables in the Structure editor, when you carry out selections by folder (see [Highlight/dim tables by folder](#)). The possible options are Dimmed (a shadow replaces the table image) and Invisible (the table disappears completely).

## Forms Page

This page lets you set the default operation and display of the 4D Form editor.

### Move

This group of options sets parameters for moving objects using the keyboard or the mouse in the Form editor.

#### Step using keyboard

This option allows setting the value (in points) of the step used for moving or resizing an object using the keyboard and the **Shift** key.

#### When moving beyond window limits

This option allows setting the behavior of the Form editor when moving an object using the mouse beyond window limits.

- **Autoscroll:** When this option is checked, this action causes the scroll of the form in the window, as if you clicked on the scroll bars. This behavior is useful for moving objects in large forms.
- **Start drag and drop:** When this option is checked, this action is interpreted as a drag and drop. The form window is not modified and the moved object can be dropped in another window (if its contents are compatible), for example, in another form. This behavior is useful for recycling objects among several forms or using object libraries (see [Creating and using custom object libraries](#)).

You can configure this option depending on your work habits and development needs.

#### Activate auto alignment by default

This option activates auto alignment by default in each new window of the Form editor. It is possible to modify this option individually in each window (refer to [Using the magnetic grid](#)).

### New form default display

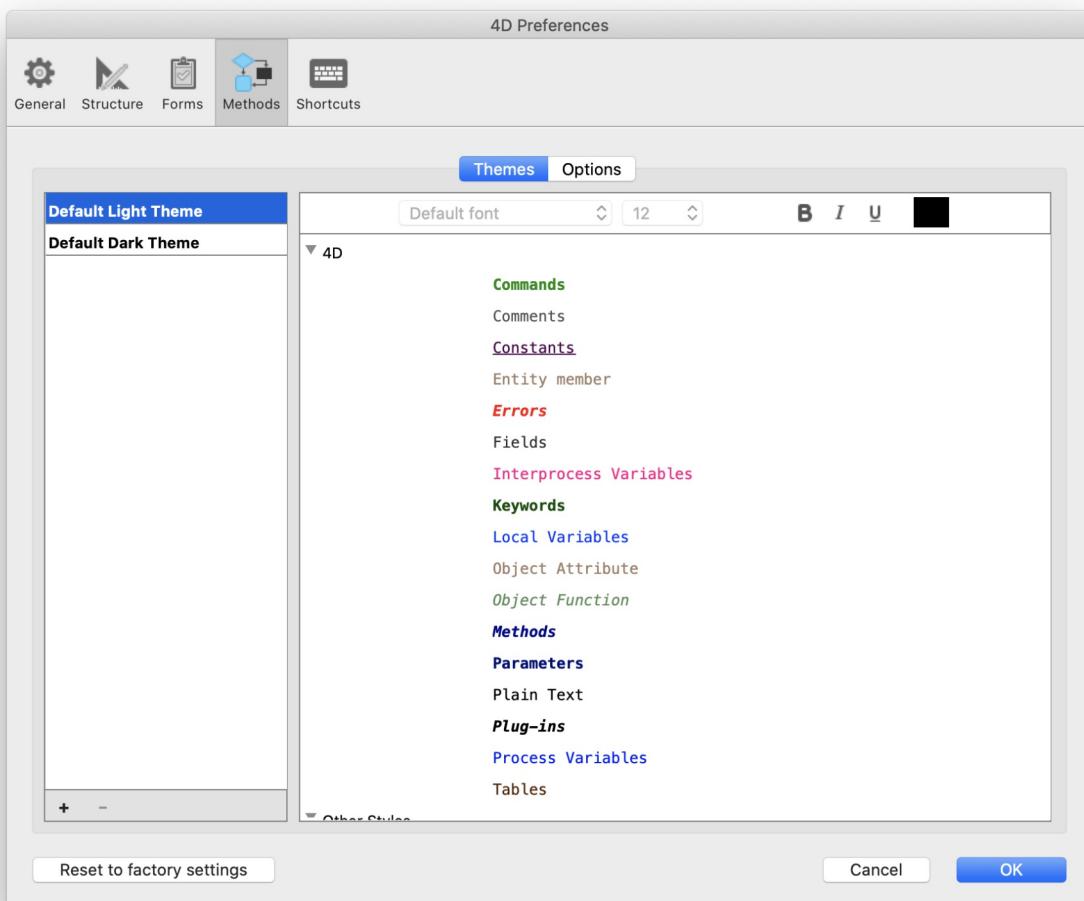
- **Limits, Rulers, ...:** check items that must be displayed by default in each new window of the Form editor. It is possible to modify the display of each window individually using the **Display** hierarchical menu of the Form editor.
- **Color for marker lines:** modifies the color of the marker lines used in the Form editor to define the different areas (header, breaks, detail and footer, etc.). For more information about markers, refer to [Using output control lines](#).
- **Default display shield:** sets which shields to display by default in each new window of the Form editor. For more information about shields, refer to [Using shields](#).

## Methods Page

This page contains parameters defining the Code Editor interface and its default display as well as options concerning its operation. It is divided into two sections accessed using the Theme and Options tabs.

### Themes

This page allows selecting, creating, or configuring Code Editor themes. A theme defines the font, font size, colors and styles of items displayed in the code editor.



### Theme list

In this list, you select the theme to apply to the code editor. All available themes are displayed, including custom themes (if any). 4D provides two themes by default:

- **Default Light Theme**
- **Default Dark Theme**

Default themes cannot be modified or deleted.

A **myTheme** theme is automatically added if you already customized Code Editor styles in previous 4D releases.

### Creating custom themes

You can create themes that you can fully customize. To create a theme, select an existing theme and click on the **+** at the bottom of the theme list. You can also add customized themes by copying theme files in the `4D Editor Themes` folder (see below).

## Custom theme files

Each custom theme is stored in a single JSON file named `themeName.json`. The JSON files for custom themes are stored in the `4D Editor Themes` folder located at the same level as the 4D [preferences file](#).

If key values are not defined in a custom theme, they default to the values from the *Default Light Theme*. If a JSON theme file is invalid, the *Default Light Theme* is loaded and an error is generated.

When a theme file is modified by an external editor, 4D must be restarted to take the modification(s) into account.

## Theme definition

Defining a theme means:

- setting a global font and font size for the whole code editor,
- assigning specific styles and colors to each 4D language element (fields, tables, variables, parameters, SQL, etc.), SQL language element (keywords, functions, etc.), and color backgrounds.

Combining different colors and styles is particularly useful for code maintenance purposes.

### Font and Font size

The **font** and **font size** menus allows you to select the font name and size used in the Code Editor entry area for all categories.

### 4D Language and SQL Language

You can set different font styles and colors (font color or background color) for each type of language element. You can select the element(s) to customize in the Category list.

### Other Styles

These options configure the various colors used in the Code Editor and debugger interfaces.

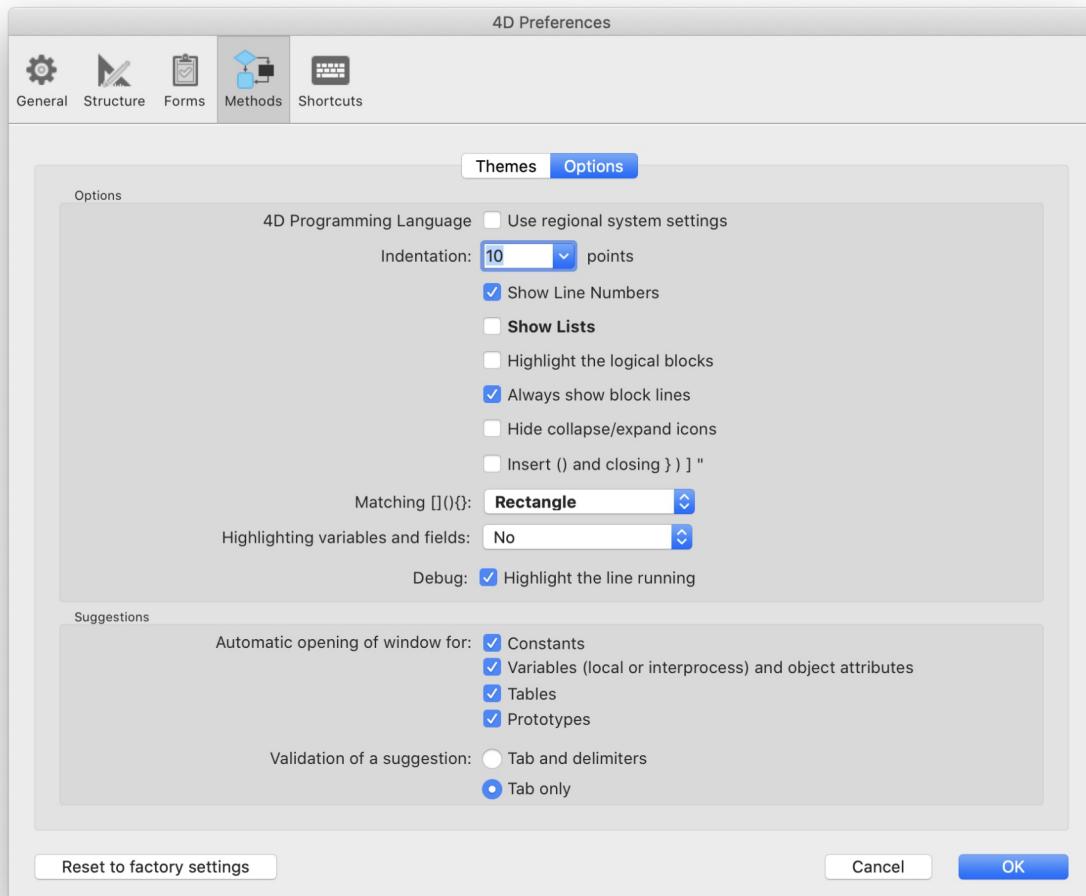
Category	Element
> 4D Language	
▼ Other Styles	
	Background color
	Cursor line background color
	Highlight of the running line in the debugger
	Execution line background color
	Border of the running line in the debugger
	Highlight background color
	Highlight of the blocks
	Highlight of the parentheses
	Highlight of the found words
	Suggested text
	Selection back color
> SQL Language	

## Description

<b>Background color</b>	Background color of Code Editor window.
<b>Border of the running line in the debugger</b>	Color of the border surrounding the line currently running in the debugger when the "Highlight line running" option is enabled in the <a href="#">Options</a> page.
<b>Cursor line background color</b>	Background color of line containing the cursor.
<b>Execution line background color</b>	Background color of line being executed in the debugger.
<b>Highlight of the found words</b>	Highlight color of words found in a search.
<b>Highlight of the parentheses</b>	Highlight color of corresponding parentheses (used when pairs of parentheses are signaled by highlighting, see <a href="#">Options</a> ).
<b>Highlight of the blocks</b>	Highlight color for selected logical blocks when the "Highlight logical blocks" option is enabled in the <a href="#">Options</a> .
<b>Highlight of the same variable or field</b>	Highlight color for other occurrences of the same variable or field text when one of the "Highlighting variables and text" option is enabled in the <a href="#">Options</a> .
<b>Highlight of the running line in the debugger</b>	Highlight color of the line currently running in the debugger when the "Highlight line running" option is enabled in the <a href="#">Options</a> .
<b>Selection back color</b>	Background color of selection.
<b>Suggested text</b>	Color of autocomplete text suggested by the Code Editor.

## Options

This page configures Code Editor display options.



## Options

### 4D Programming Language (Use regional system settings)

Allows you to disable/enable the "international" code settings for the local 4D application.

- **unchecked** (default): English-US settings and the English programming language are used in 4D methods.
- **checked**: Regional settings are used in 4D methods.

If you modify this option, you need to restart the 4D application so that the change is taken into account.

### Indentation

Changes the indentation value for the 4D code in the Code Editor. The width must be specified in points (10 by default).

4D code is automatically indented in order to reveal its structure:

```

1  If ($vItemPos#0)
2      // Get the list item information
3      GET LIST ITEM(hList;$vItemPos;$v
4          // Is the item a Department item
5      If ($vItemRef ?? 31)
6          // If so, it is a double-click
7          ALERT("You double-clicked on the
8      Else
9          // If not, it is a double-click
10         // Using the parent item ID to
11         SvlDepartmentID:=List item parent
12         QUERY([Departments];[Departments]
13             // Tell where the Employee is v
14             ALERT("You double-clicked on the
15         End if
16     End if
17
18 standard indentation

```

Modifying this default value can be useful if your methods contain complex algorithms with many levels of embedding. Narrower indentation can be used in order to limit horizontal scrolling.

## Show Line Numbers

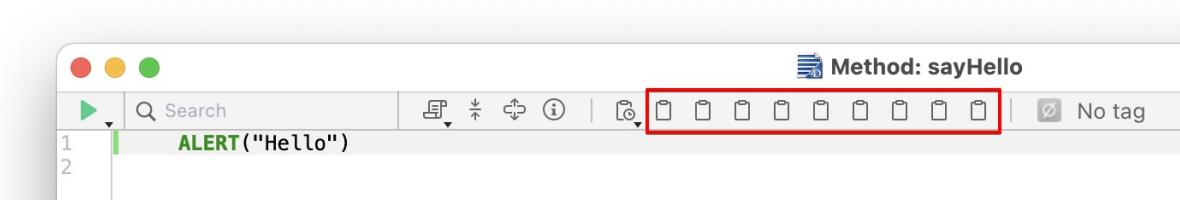
Lets you display the line numbers by default in each window of the Code Editor. You can also show/hide line numbers for the current window directly from the Code Editor.

## Show Lists

Lets you choose whether or not to show the lists of objects (Commands, Tables and fields, etc.) by default when the Code Editor window is opened. You can also show or hide each list directly from the Code Editor.

## Show clipboards

Lets you choose whether or not to show the multiple clipboards in the code editor.



The corresponding [clipboard shortcuts](#) are still active when these clipboards are hidden.

## Highlight the logical blocks

When checked, the whole code belonging to a logical block (If/End if for example) is highlighted when the mouse is placed over the expanded node:

```

12  El If (<>PS_EditMovies=0)
13      | L <>PS_EditMovies:=New process ($CurrentMethName;
14      El
15          R BRING TO FRONT(<>PS_EditMovies)
16          End if
17

```

The highlight color can be set in the [Theme](#) page.

## Always show block lines

Allows to hide vertical block lines permanently. The block lines are designed to visually connect nodes. By default, they are always displayed (except when collapse/expand icons are hidden, see below).

```

9  □ If (Count
10
11      $CurrentM
12      □ If (<>PS_
13          <>PS_Ed:
14      □ Else
15          BRING TC
16      End if
17
18  □ Else
19

```

## Hide collapse/expand icons

Allows you to hide all expand/collapse icons by default when displaying code. When the option is checked, node icons (as well as local block lines, see above), are displayed temporarily when the mouse is placed over a node:

```

9  If (Count
10
11      $CurrentM
12      If (<>PS_
13          <>PS_Ed:
14      Else
15          BRING TC
16      End if
17
18  Else
19

```

## Insert () and closing } ) ] "

Enables automatic insertion of () and closing braces while typing code. This option controls two automatic features:

- **parentheses pair ()**: Added after a 4D command, keyword or project method inserted from a suggestion or completion list, if the inserted element requires one or more mandatory arguments. For example, if you type "C\_OB" and press Tab, 4D writes "C\_OBJECT()" and sets the insertion point inside the ().
- **closing }, ), ], or "**: Character added when you type respectively an opening {, (, ], or ". This feature allows inserting matching pairs of symbols at the insertion point or surrounding a selected text. For example, if you highlight a string and type a single ", the whole selected string will be enclosed in ":".

```
Hello_world
"Hello_world"
```

## Matching [](){}]

Sets the graphic signaling of matching braces in the code. This signaling appears whenever a square bracket, parenthesis, or curly bracket is selected. The following options are available:

- **None**: No signaling
- **Rectangle** (default): Braces surrounded by a black line

```
INSERT MENU ITEM(main bar;-1;Get indexed string(79;1);FileMenu)
```

- **Background Color:** Braces highlighted (the color is set in the [Theme](#) page).
- **Bold:** Braces displayed in bold.

## Highlighted variables and fields

Allows to highlight all occurrences of the same variable or field in an open method window.

```
4   C_LONGINT(<>PS_EditMovies)
5   C_LONGINT($Window)
6   C_TEXT($CurrentMethName)
7   C_LONGINT($Win)
8
9   If (Count parameters=0)
10
11   $CurrentMethName:=Current method
12   If (<>PS_EditMovies=0)
13       <>PS_EditMovies:=New process ($C
14   Else
15       BRING TO FRONT(<>PS_EditMovies)
16   End if
```

- **No**(default): No highlight
- **On cursor:** All occurrences are highlighted when the text is clicked
- **On selection:** All occurrences are highlighted when the text is selected

The highlight color can be set in the [Theme](#) page.

## Debug (Highlight the line running)

Highlights the line that is currently running in the debugger in addition to the regular yellow arrow indicator.

```
// Create a new empty
→ hlList:=New list
// Select all the rec
```

If you deselect this option, only the yellow arrow is shown.

## Suggestions

This area lets you configure autocomplete mechanisms in the Code Editor to adapt it to your own work habits.

## Description

Triggers the automatic display of the suggestion window for:

- Constants
- Variables (local and interprocess) and object attributes
- Tables
- Prototypes (*i.e.*, class functions)

For example, when the "Variables (local or interprocess) and object attributes" option is checked, a list of suggestions appears when you type

Automatic the \$ character:

opening of

window    **GET LIST ITEM(\$**

for



You can disable this functioning for certain elements of the language by deselecting their corresponding option.

Sets the entry context that allows the Code Editor to validate automatically the current suggestion displayed in the autocomplete window.

- **Tab and delimiters**

When this option is selected, you can validate the current selection with the Tab key or any delimiter that is relevant to the context. For example, if you enter "ALE" and then "(", 4D automatically writes "ALERT(" in the editor. Here is the list of delimiters that are taken into account:

( ; : = < [ {

- **Tab only**

When this option is selected, you can only use the Tab key to insert the current suggestion. This can be used more particularly to facilitate the entry of delimiter characters in element names, such as \${1}.**Note:** You can also double-click in the window or press the Carriage return key to validate a suggestion.

Validation  
of a  
suggestio  
n for

## Shortcuts Page

This page lists all the shortcuts used in the 4D Design environment (except for standard "system" shortcuts, such as Ctrl+C/Command+C for the Copy command).

To modify a shortcut, you can select/deselect the item to modify (Shift, Alt or letter key) in the list. You can also double-click on a shortcut to configure it using a specific dialog box.

Note that each shortcut implicitly includes the **Ctrl** (Windows) or **Command** (macOS) key.

If you edit this list, your custom shortcuts settings are stored in a *4DShortcutsXX.xml* file, created at the same level as the [user preferences file](#). Hence, each time 4D is updated your keyboard shortcut preferences remain.

# Administration

How to monitor your 4D applications

## 4D Server Administration Window

9 items

## Web Administration

2 items

## Command Line Interface

You can use the macOS Terminal or the Windows console to drive your 4D applicatio…

## TLS Protocol

All 4D servers can communicate in secured mode through the TLS (Transport Layer S…

## Managing 4D Licenses

Once installed on your disk, you must activate your 4D products in order to be able t…

## MSC

9 items

## Backup and Restore

4 items

## **Data Collection**

To help us make our products always better, we automatically collect data regarding ...

## 4D Server Administration Window

When 4D Server is launched with interface under Windows or macOS, a graphical administration window is available, providing many analysis and control tools for the published 4D application. To display the 4D Server Administration window for the opened project, select the **Window > Administration** menu item, or press **Ctrl+U**.

The 4D Server administration window can be accessed from a remote 4D.

For more information about this point, please refer to the [Administration from Remote Machines](#) page.

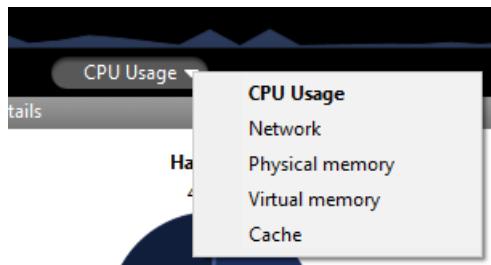
## Monitor Page

The **Monitor** page displays dynamic information concerning database use as well as information about the system and the 4D Server application.

On Windows, some of the system information displayed on this page are retrieved via the Windows "Performance Analyzer" tools. These tools can only be accessed when the user that opened the session where 4D Server was launched has the necessary administration authorization.

## Graphic area

The graphic area lets you see the evolution in real time of several parameters: the CPU usage, network traffic and memory. You select the parameter to be displayed via a menu found in the center of the window:



- **CPU Usage:** Overall CPU usage of the machine, for all applications taken together. The specific part of 4D Server in this usage rate is provided in the "Processors" information area.
- **Network:** Number of bytes received per second by the machine (server or client). The number of bytes sent is provided in the "Network" information area.
- **Physical memory:** Quantity of RAM memory of machine used by 4D Server. A more detailed view of memory use is provided in the "Memory" information area.
- **Virtual memory:** Quantity of virtual memory used by the 4D Server application. This memory is allocated by the system according to the application needs. The value found at the bottom right of the area indicates the quantity of memory currently being used. The value found at the top left indicates the maximum quantity of usable virtual memory. The maximum value is calculated dynamically according to the general memory settings of the application.
- **Cache:** Quantity of cache memory used by the 4D Server application. The value found at the bottom right of the area indicates the quantity of memory currently being used. The value found at the top left indicates the total size of the cache memory, as set via the Settings.

Note that when this option is selected, the graph area scrolling is slowed down since an efficient analysis of the cache is generally carried out over a fairly long observation period.

## Overview Area

The "Overview" area provides various information concerning the system, application and licenses installed on the 4D Server machine.

- **System Information:** Computer, system and IP address of server
- **Application Information:** Internal version number of 4D Server and Volume

Shadow Copy status

- **Maximum connections:** Number of simultaneous connections allowed by type of server
- **License:** Description of license. When the product license or one of its attached expansions expires in less than 10 days, e.g. in case of a subscription-license, 4D Server tries to automatically renew the license from the 4D user account. In this case, if the automatic renewal failed for some reason (connection error, invalid account status, non-prolongated contract...), a warning icon is displayed next to the license to alert the server administrator. Additional information about the license renewal status can be displayed in a tip when you hover the mouse over the area:

Usually, you will need to check the [Licences Manager](#).

## Details Area

The "Details" area repeats part of the information displayed in the graphic area and provides additional information as well.

- **Hard drive:** Overall capacity of the hard disk and distribution of the space used by the database data (data file + data index), the space used by other files and the free space available.
- **Memory:** RAM memory installed on the machine and amount of memory used by 4D Server, by other applications or that is free. The memory used by 4D Server can also be displayed dynamically in the graphic area.
- **Processors:** Instant occupancy rate for processor(s) of the machine by 4D Server and by other applications. This rate is constantly recalculated. The occupancy rate by 4D Server can also be displayed dynamically in the graphic area.
- **Network:** Instantaneous number of bytes sent and received by the machine (server or client). This value is updated constantly. The number of bytes received by can also be displayed dynamically in the graphic area.

## Users Page

The **Users** page lists the 4D users connected to the server.

The "Users" button indicates, in parentheses, the total number of users connected to the server (this number does not take into account any display filters applied to the window). The page also contains a dynamic search area and control buttons. You can modify the order of the columns by dragging and dropping their header areas.

You can also sort the list of column values by clicking on its header. Click several times to specify in turn an ascending/descending order.



## List of Users

For each user connected to the server, the list provides the following information:

- System of the client machine (macOS or Windows) as an icon.
- **4D User:** Name of the 4D user, or alias if set with the [SET USER ALIAS](#) command on the user machine. If passwords are not activated and no alias has been set, all users are named "Designer".
- **Machine name:** Name of the remote machine.
- **Session name:** Name of the session opened on the remote machine.
- **IP Address:** IP address of the remote machine.
- **Login date:** Date and time of the remote machine connection.
- **CPU Time:** CPU time consumed by this user since connecting.
- **Activity:** Ratio of time that 4D Server devotes to this user (dynamic display). "Sleeping" if the remote machine has switched to sleep mode (see below).

## Managing sleeping users

4D Server specifically handles cases where a machine running a 4D remote application switches to sleep mode while its connection to the server machine is still active. In this case, the connected 4D remote application automatically notifies 4D Server of its imminent disconnection. On the server, the connected user changes to a **Sleeping** activity status:

This status frees up resources on the server. In addition, the 4D remote application reconnects to 4D Server automatically after waking up from sleep mode.

The following scenario is supported: a remote user stops working for awhile, for example during a lunch break, but keeps the connection to the server open. The machine switches to sleep mode. When the user returns, they wake the machine up and the 4D remote application automatically recovers its connection to the server as well as the session context.

A sleeping remote session is automatically dropped by the server after 48 hours of inactivity. You can modify this default timeout using the [SET DATABASE PARAMETER](#) command with the `Remote connection sleep timeout` selector.

## Search/filtering Area

This feature can be used to reduce the number of rows displayed in the list to those that correspond to the text entered in the search area. The area indicates the columns where the search/filtering will be carried out. On the Users page, it will be the 4D User, Machine name and Session name columns.

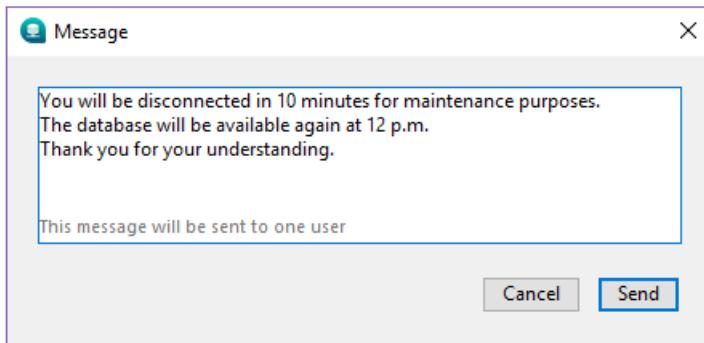
The list is updated in real time as you enter text in the area. It is possible to enter more than one value to be searched for: separate the values with a semi-colon. The OR type operator is used in this case. For example, if you enter "John;Mary;Peter," only rows with John OR Mary OR Peter in the target columns will be kept.

## Administration Buttons

This page includes three control buttons. These are active if at least one row is selected. You can select several rows by holding down the **Shift** key for an adjacent selection or **Ctrl** (Windows) / **Command** (macOS) key for a non-adjacent selection.

### Send message

This button can be used to send a message to the 4D users selected in the window. If no user is selected, the button is not active. When you click on this button, a dialog box appears that lets you enter the message. The dialog box indicates the number of users that will receive this message:



The message will be displayed as an alert on the remote machines.

You can perform the same action for remote users with the [SEND MESSAGE TO REMOTE\\_USER](#) command.

### Watch Processes

This button can be used to directly show the processes of the user(s) selected on the [Processes page](#) of the window. When you click on this button, 4D Server switches to the Processes page and enters the selected user names in the search/filtering area.

### Drop user

This button can be used to force the selected user(s) to disconnect. When you click on this button, a warning dialog box appears so that you can confirm or cancel this operation (hold down **Alt** key while clicking on the **Drop user** button to disconnect the selected user(s) directly without displaying the confirmation dialog box).

You can perform the same action for remote users with the [DROP\\_REMOTE\\_USER](#) command.



## Processes Page

The **Processes** page lists all the processes underway.

The "Processes" button indicates, in parentheses, the total number of processes running in the server (this number does not take into account any display filters applied to the window nor the state of the **Display processes by groups** option).

You can change the order of the columns by simply dragging and dropping the column header areas. You can also sort the list of column values by clicking on its header.

Like the Users page, this page contains a dynamic [search/filtering area](#) that can be used to reduce the number of rows displayed in the list to those that correspond to the text entered in the search area. The search/filtering is carried out in the Session and Process name columns.

There are also three shortcut buttons that can be used to filter by the type of process displayed in the window:



- **Users processes:** Processes generated by and for the user sessions. These processes are preceded by an icon in the form of a figure.
- **4D Processes:** Processes generated by the 4D Server engine. These processes are preceded by an icon in the form of a notched wheel.
- **Spare processes:** Processes that are inactive but kept temporarily and that can be reused at any time. This mechanism optimizes the reactivity of 4D Server. These processes are preceded by an icon in the form of a dimmed figure.

The **Display processes by groups** option lets you group together the internal processes of 4D Server as well as the client processes, for better readability. When you check this option:

- the "twinned" 4D client processes (main 4D client process and 4D client base process, see [Process Type](#)) are grouped as one,
- a "Task managers" group is created; it includes the internal processes dedicated to dividing up tasks (Shared balancer, Net session manager, Exclusive pool worker),
- a "Client managers" group is created; it includes various client internal processes.

The lower area of the window is used to display the graphic representation of the activity of the selected process(es).

You can select several rows by holding down the **Shift** key for an adjacent selection or **Ctrl** (Windows) / **Command** (macOS) for a non-adjacent selection.

The activity of the process is the percentage of time that 4D Server has devoted to this process (ratio). The window provides the following information for each process:

- Type of process (see below),
- Session/Info:
  - 4D process - blank,
  - User process - 4D user name,

- Web process - URL path,
- Name of the process,
- Number of the process (as returned by the [New process](#) command for example).  
The process number is the number assigned on the server. In the case of a global process, this number may be different from that assigned on the client machine.
- Current state of the process,
- Running time (in seconds) of the process since its creation,
- Percentage of time that 4D Server has devoted to this process (ratio).

## Process Type

Each process is identified by an icon as well as a type. The color and form of the icon indicates the type of process:

icon	type
	Application server
	SQL Server
	DB4D Server (database engine)
	Web Server
	SOAP Server
	Protected 4D client process (development process of a connected 4D)
	Main process of a connected 4D client or process created with <a href="#">New process</a> on a
	connected 4D client (Collaborative process, equivalent on the server of the process created on the client machine)
	4D client base process (process parallel to a 4D client process. Preemptive process responsible for controlling the corresponding main 4D client process)
	Spare process (former or future "4D client database process")
	SQL server worker process
	HTTP server worker process
	Stored procedure (process launched by a connected 4D and running on the server)
	Web method (launched by a 4DACTION for example)
	Web method (preemptive)
	SOAP method (launched by a Web Service)
	SOAP method (preemptive)
	Logger
	TCP connection listener
	TCP session manager
	Other process
	Worker process (cooperative)
	4D client process (preemptive)
	Stored procedure (preemptive process)
	Worker process (preemptive)

Each main 4D client process and its "twinned" 4D client base process are grouped together when the **Display processes by groups** option is checked.

## Administration Buttons

The page also has five control buttons that act on the selected process(es). Note that only user processes can be acted upon.



- **Abort Process:** can be used to abort the selected process(es). When you click on this button, a warning dialog box appears so that you can confirm or cancel the operation.

You can also abort the selected process(es) directly without displaying the confirmation dialog box by holding down the **Alt** key while clicking on this button, or by using the [ABORT PROCESS BY ID](#) command.

- **Pause Process:** can be used to pause the selected process(es).
- **Activate Process:** can be used to reactivate the selected process(es). The processes must have been paused previously (using the button above or by programming); otherwise, this button has no effect.
- **Debug Process:** can be used to open on the server machine one or more debugger windows for the selected process(es). When you click on this button, a warning dialog box appears so that you can confirm or cancel the operation. Note that the debugger window is only displayed when the 4D code is actually executed on the server machine (for example in a trigger or the execution of a method having the "Execute on Server" attribute).

You can also debug a process directly without displaying the confirmation dialog box by holding down the **Alt** key while clicking on this button.

- **Watch users:** used to display, on the [Users page](#), all the processes of the selected user(s). This button is active when at least one user process is selected.

## Maintenance Page

The **Maintenance** page of the 4D Server Administration window provides information concerning the current operation of the application. It also provides access to basic maintenance functions:

### Last verification/compacting

These areas indicate the date, time and status of the last [data verification](#) and [compacting operation](#) carried out on the database.

#### Verify Records and Indexes

This button can be used to launch the verification operation directly, without interrupting the server. Note that the server may be noticeably slowed down during the operation.

All the records and all the indexes of the database are verified. If you want to be able to target the verification or have additional options available, you will need to use the [Maintenance and Security Center](#) (MSC).

After verification, a report file is generated in XML format on the server in the [maintenance Logs](#) folder. The **View Report** button (named **Download Report** if the operation was carried out from a remote machine) lets you display the file in your browser.

#### Compact Data...

Thus button can be used to launch a data compacting operation directly. This operation requires stopping the server: when you click on this button, the 4D Server shutdown dialog box appears so that you can choose how to interrupt the operation:

After the actual interruption of the application service, 4D Server carries out a standard compacting operation on the database data. If you want to have additional options available, you will need to use the [MSC](#).

Once the compacting is finished, 4D Server automatically restarts the application. The 4D users can then be reconnected.

If the request for compacting was carried out from a remote 4D remote machine, this machine is automatically reconnected by 4D Server.

After verification, a report file is generated in XML format on the server in the [maintenance Logs](#) folder. The **View Report** button (named **Download Report** if the operation was carried out from a remote machine) lets you display the file in your browser.

### Uptime

This area indicates the duration of the 4D Server application execution since the last time it was started (days, hours and minutes).

#### Restart server...

This button can be used to immediately close and restart the project. When you click on this button, the 4D Server shutdown dialog box appears so that you can choose how to interrupt the operation. After validation, 4D Server automatically closes and reopens the project. The 4D users can then be reconnected.

If the request for restarting was carried out from a remote 4D machine, this machine is automatically reconnected by 4D Server.

## Last backup

This area indicates the date and time of the [last backup](#) of the database and provides information about the next scheduled automatic backup (if any). Automatic backups are configured using the **Scheduler** page of the structure settings.

- **Last backup:** date and time of last backup.
- **Next backup:** date and time of next scheduled backup.
- **Needed space:** estimated space needed for the backup. The actual size of the backup file may vary according to the settings (compression, etc.) and according to variations of the data file.
- **Available space:** space available on the backup volume.

The **Start backup** button can be used to backup the database immediately using the current backup parameters (files backed up, location of archives, options, etc.). You can view these parameters by clicking on the **Settings...** button. During a backup on the server, the client machines are "blocked" (but not disconnected) and it is not possible for any new clients to connect.

## Request and Debug logs

This area indicates the server log files recording duration (when log files are activated) and allows you to control their activation.

Refer to the [Description of log files](#) section for details on log files.

### Start/Stop Request and Debug Logs

The **Start Request and Debug Logs** button starts log files. Since this may noticeably deteriorate server performance, it is to be reserved for the development phase of the application.

This button only logs operations that are executed on the server.

When the logs have been activated, the button title changes to **Stop Request and Debug Logs**, so that you can stop recording requests at any time. Pay attention to the fact that restarting the log after stopping it "erases" the previous file.

### View Report

The **View Report** button (named **Download report** if the operation was carried out from a remote desktop client) lets you open a system window displaying the request log file.

### Load logs configuration file

This button allows you to load a special server [log configuration file](#) (`.json` file). Such a file can be provided by 4D technical services to monitor and study specific cases.

### Pause logging

This button suspends all currently logging operations started on the server. This feature can be useful to temporarily lighten the server tasks.

When the logs have been paused, the button title changes to **Resume logging**, so that you can resume the logging operations.

You can pause and resume logging using the [SET DATABASE PARAMETER](#) command.

## Application Server Page

The Application Server page groups together information about the desktop application published by 4D Server and can be used to manage this publication.

The upper part of the page provides information about the current status of the 4D Server application server.

- **State:** Started or Stopped.
- **Starting time:** Date and time the application server was launched. This date corresponds to the opening of the project by 4D Server.
- **Uptime:** Time elapsed since last opening of the project by the server.

## Accept/Reject New Connections

This button toggles and can be used to manage the access of new desktop client machines to the application server.

By default, when the project is published:

- The button is titled "Reject new connections."
- New desktop clients can connect freely (within the limit of the connections permitted by the license).
- The project name is published in the remote connection dialog box (if the "At Startup Publish Database Name in the Connection Dialog" option is checked in the Preferences).

If you click on the **Reject new connections** button:

- The button title changes to "Accept new connections."
- No new desktop client can then connect. Clients attempting to connect will receive the following message:
  - The project name no longer appears in the remote connection dialog box.
  - Desktop clients that are already connected are not disconnected and can continue to work normally.

You can perform the same action with the [REJECT NEW REMOTE CONNECTIONS](#) command.

- If you click on the **Accept new connections button**, the application server returns to its default state.

This feature permits, for example, an administrator to carry out various maintenance operations (verification, compacting, etc.) just after having started the server. If the administrator uses a remote connection, they can be certain to be the only one modifying the data. It is also possible to use this function in preparation of a maintenance operation which requires that there be no desktop client machine connected.

## Information

## Configuration

This area provides information about the 4D project published by the server: name and location of data and structure files and name of database log file. You can click on the structure or data file name in order to view its complete pathname.

The **Mode** field indicates the current execution mode of the application: compiled or interpreted.

The lower part of the area indicates the server configuration parameters (launched as service, port and IP address) and the enabling of TLS for client-server connections (does not concern SQL nor HTTP connections).

## Memory

This area indicates the **Total cache memory** (parameter set in the settings) and the **Used cache memory** (dynamic allocation by 4D Server according to its needs).

## Application Server Connections

- **Maximum:** maximum number of simultaneous client connections allowed for the application server. This value depends on the license installed on the server machine.
- **Used:** actual number of connections currently being used.

## SQL Server Page

The **SQL Server** page groups together information about the integrated SQL server of 4D Server. It also includes a button that can be used to control the activation of the server.

The upper part of the page provides information about the current status of the SQL server of 4D Server.

- **State:** Started or Stopped
- **Starting time:** Date and time the SQL server was last launched.
- **Uptime:** Time elapsed since last startup of the SQL server.

## Start / Stop SQL Server

This button toggles and can be used to control the activation of the 4D Server SQL server.

- When the SQL server state is "Started," the button is titled **Stop SQL Server**. If you click on this button, the 4D Server SQL server is immediately stopped; it no longer replies to any external SQL requests received on the designated TCP port.
- When the SQL server state is "Stopped," the button is titled **Start SQL Server**. If you click on this button, the 4D Server SQL server is immediately started; it replies to any external SQL queries received on the designated TCP port. Note that you will need a suitable license to be able to use the 4D SQL server.

The SQL server can also be launched automatically on application startup (option in the Settings) or by programming.

## Information

### Configuration

This area provides information about the SQL server configuration parameters: automatic launching on startup, listening IP address, TCP port (19812 by default) and enabling of SSL for SQL connections (does not concern 4D nor HTTP connections).

These parameters can be modified via the 4D Settings.

### Connections

Number of SQL connections currently open on 4D Server.

### Maximum Connections

Maximum number of simultaneous SQL connections allowed. This value depends on the license installed on the server machine.

## HTTP Server Page

The **HTTP Server** page groups together information about the operation of the Web server and SOAP server of 4D Server. The Web server lets you publish Web content such as HTML pages or pictures for Web browsers, and to handle REST requests. The SOAP server manages the publication of Web Services. These servers rely on the internal HTTP server of 4D Server.

The upper part of the page provides information about the current status of the HTTP server of 4D Server.

- **State:** Started or Stopped
- **Starting time:** Date and time the HTTP server was last launched.
- **Uptime:** Time elapsed since last startup of the HTTP server.
- **Total HTTP hits:** Number of (low level) HTTP hits received by the HTTP server since it was started.

## Start/Stop HTTP Server

This button toggles and can be used to control the activation of the 4D Server HTTP server.

- When the HTTP server state is "Started," the button is titled **Stop HTTP Server**. If you click on this button, the 4D Server HTTP server is immediately stopped; the Web server, REST server, and SOAP server no longer accept any requests.
- When the HTTP server state is "Stopped," the button is titled **Start HTTP Server**. If you click on this button, the 4D Server HTTP server is immediately started; Web, REST, and SOAP requests are accepted.

You must have a suitable license in order to be able to start the HTTP server.

The HTTP server can also be launched automatically on application startup (Settings) or by programming.

## Web Information

This area provides specific information about the Web server of 4D Server.

- **Web requests:** Accepted or Rejected. This information indicates whether the Web server is activated. Since the Web server is directly linked to the HTTP server, Web requests are accepted when the HTTP server is started and rejected when it is stopped.
- **Maximum connections:** Maximum number of Web connections allowed. This value depends on the license installed on the server machine.

## SOAP Information

This area provides specific information about the SOAP server of 4D Server and includes a control button.

- **SOAP requests:** Accepted or Rejected. This information indicates whether the SOAP server is activated. In order for SOAP requests to be accepted, the HTTP server must be started and the SOAP server must explicitly accept the requests (see the Accept/Reject button).

- **Maximum connections:** Maximum number of SOAP connections allowed. This value depends on the license installed on the server machine.
- **Accept/Reject SOAP requests** button: This button toggles and can be used to control the activation of the 4D Server SOAP server. This button modifies the value of the **Allow Web Services Requests** option on the "Web Services" page of the Settings (and vice versa). You can also use the [SOAP REJECT NEW REQUESTS](#) command to refuse new SOAP requests, however this does not modify the value of the **Allow Web Services Requests** option.

If you click on the **Accept SOAP requests** button and the HTTP server is stopped, 4D automatically starts it.

## HTTP Server Configuration

This area provides information about the configuration parameters and operation of the HTTP server:

- **Auto-launched at startup:** parameter set via the Settings.
- **HTTP Server processes (used/total):** number of HTTP processes created on the server (current number of processes / total of all processes created).
- **Cache memory:** size of HTTP server cache memory, when it is activated (size actually used by cache / maximum size theoretically allocated to the cache in the Settings). You can click on the **Clear Cache** button to empty the current cache.
- **Listening to IP, HTTP Port** (80 by default), **TLS enabled** for HTTP connections (does not concern 4D nor SQL connections) and **HTTPS Port** used: current [configuration parameters](#) of the HTTP server, specified through the Settings or by programming.
- **Log file information:** name, format and date of the next automatic log backup of the HTTP server (logweb.txt file).

## Real Time Monitor Page

The Real Time Monitor page monitors the progress of "long" operations performed by the application in real time. These operations are, for example, sequential queries, execution of formulas, etc.

This page is available in the administration window of the server machine and also from a remote 4D machine. In the case of a remote machine, this page displays data from operations performed on the server machine.

A line is added for each long operation performed on the data. This line automatically disappears when the operation is complete (you can check the **Display operations at least 5 seconds** option to keep quick operations on screen for 5 seconds, see below).

The following information is provided for each line:

- **Start Time:** starting time of operation in the format: "dd/mm/yyyy - hh:mm:ss"
- **Duration (ms):** duration in milliseconds of operation in progress
- **Information:** title of operation.
- **Details:** this area displays detailed information which will vary according to the type of operation selected. More specifically:
  - **Created on:** indicates whether the operation results from a client action (Created on client) or if it was started explicitly on the server by means of a stored procedure or the "Execute on server" option (Created on server).
  - **Operation Details:** Operation type and (for query operations) query plan.
  - **Sub-operations (if any):** Dependent operations of the selected operation (e.g. deleting related records before a parent record).
  - **Process Details:** Additional information concerning the table, field, process or client, depending on the type of operation

Real-time monitoring page uses the [GET ACTIVITY SNAPSHOT](#) command internally. More information can be found in this command description.

The page is active and updated permanently as soon as it is displayed. It should be noted that its operation can significantly slow the execution of the application. It is possible to suspend the updating of this page in one of the following ways:

- clicking on the **Pause** button,
- clicking in the list,
- pressing the space bar.

When you pause the page, a "PAUSED" message appears and the button label changes to **Resume**. You can resume monitoring of the operations by performing the same action as for pausing.

## Advanced mode

The RTM page can display additional information, if necessary, for each listed operation.

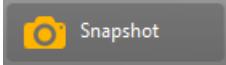
To access the advanced mode for an operation, press **Shift** and select the desired operation. All available information is then displayed in the "Process Details" area without any filtering (as returned by the [GET ACTIVITY SNAPSHOT](#) command). Available information depends on the operation selected.

Here is an example of information displayed in standard mode:

In advanced mode (**Shift+Click** on the operation), additional information is displayed:

## Snapshot button

The **Snapshot** button allows you to copy to the clipboard all the operations displayed in the RTM panel, as well as their related details (process and sub-operation info):



## Display operations at least 5 seconds

If you check the **Display operations at least 5 seconds** option, any listed operation will be displayed on the page for at least five seconds, even after its execution is finished. Retained operations appear dimmed in the operation list. This feature is useful for getting information about operations that execute very quickly.

## Administration from Remote Machines

You can administer the 4D Server application from a remote 4D (client machine) by opening the 4D Server administration window on the client machine.

### Opening the administration window on a remote 4D machine

To open a server administration window from a client machine, you must be connected to the remote database as a Designer or Administrator. Otherwise, when you attempt to open the administration window, a privilege error (-9991) is generated.

This window can be accessed in one of two manners:

- Choose the **Administration Window** command from the **Help** menu or click on the corresponding button in the 4D tool bar.
- Execute the `OPEN ADMINISTRATION WINDOW` command.

A [server administration window](#) then appears on the client machine.

### Specificities of administration via a remote 4D machine

A client machine displaying the server administration window has access to all the available information and can act upon the processes and the starting/stopping of servers. When the server administration window is displayed on a remote machine, there are nevertheless certain restrictions and specific features concerning its operation:

- On the [Process page](#), it is not possible to debug a user process (since the debug window appears on the server machine).
- On the [Maintenance Page](#), it is possible to execute actions that cause all the clients to be disconnected and the server to be restarted (compacting and restarting operations). In this case, the client machine requesting the operation is automatically reconnected on restarting.
- On the [Maintenance Page](#), the **View Report** buttons are renamed **Download Report** after the execution of a maintenance operation. These files are downloaded into the local database folder on the client machine before being displayed.

# Web Administration

4D web tools for administrating and monitoring your applications.

## [WebAdmin](#)

An embedded web server component, named WebAdmin, is used by 4D and 4D Server to provide a web interface for managing your application.

## [Data Explorer](#)

The Data Explorer provides a web interface to view and query data in your project database.

## WebAdmin

An embedded web server component, named `WebAdmin`, is used by 4D and 4D Server to provide a secured web access to specific management features such as the [Data Explorer](#). You can connect locally or remotely to this web server from a browser or any web application and access the associated 4D application.

The WebAdmin handles the authentication of users with "WebAdmin" privileges, so that they can open administration sessions and access dedicated interfaces.

This feature can be used in 4D applications running headless as well as 4D applications running with interfaces.

## Starting the WebAdmin web server

By default, the `WebAdmin` web server is not launched. You need to configure the launch at startup, or (in versions with interface) launch it manually using a menu item.

### Launch at startup

You can configure the `WebAdmin` web server to be launched at 4D or 4D Server application startup (before any project is loaded).

- If you use a 4D application with interface, select the **File > Web Administration > Settings...** menu item.

Check the **Web server administration automatic startup** option in the settings dialog box:

- Whether you use 4D application which is headless or not, you can enable the automatic startup mode using the following *Command Line Interface* argument:

```
open ~/Desktop/4D.app --webadmin-auto-start true
```

If the TCP port used by the `WebAdmin` web server ([HTTPS](#) or [HTTP](#), depending on the settings) is not free at startup, 4D will try successively the 20 following ports, and use the first one that is available. If no port is available, the web server is not launched and an error is displayed or (headless application) logged in the console.

### Start and stop

If you use a 4D application with interface, you can start or stop the `WebAdmin` web server for your project at any moment:

Select the **File > Web Administration > Start Server** menu item.

The menu item becomes **Stop Server** when the server is launched; select **Stop Server** to stop the `WebAdmin` web server.

## WebAdmin Settings

Configuring the `WebAdmin` component is mandatory in particular to define the [access key](#). By default when the access key is not set, access via a URL is not allowed.

You can configure the `WebAdmin` component using the [Web Administration settings dialog box](#) (see below).

If you use a headless 4D application, you can use [Command Line Interface arguments](#) to define basic settings. You will have to customize the settings file to define advanced parameters.

### Settings dialog box

To open the Web Administration settings dialog box, select the **File > Web Administration > Settings...** menu item.

The following dialog box is displayed:

#### Web server administration automatic startup

Check this option if you want the `WebAdmin` web server to be automatically launched when the 4D or 4D Server application starts ([see above](#)). By default, this option is not checked.

#### Accept HTTP connections on localhost

When this option is checked, you will be able to connect to the `WebAdmin` web server through HTTP on the same machine as the 4D application. By default, this option is checked.

#### Notes:

- Connections with HTTP other than localhost are never accepted.
- Even if this option is checked, when [Accept HTTPS](#) is checked and the TLS configuration is valid, localhost connections use HTTPS.

#### HTTP Port

Port number to use for connections through HTTP to the `WebAdmin` web server when the **Accept HTTP connections on localhost** option is checked. Default value is 7080.

#### Accept HTTPS

When this option is checked, you will be able to connect to the `WebAdmin` web server through HTTPS. By default, this option is checked.

#### HTTPS Port

Port number to use for connections through HTTPS to the `WebAdmin` web server when the **Accept HTTPS** option is checked. Default value is 7443.

#### Certificate folder path

Path of the folder where the TLS certificate files are located. By default, the certificate folder path is empty and 4D or 4D Server uses the certificate files embedded in the 4D application (custom certificates must be stored next to the project folder).

## Debug log mode

Status or format of the HTTP request log file (HTTPDebugLog\_nn.txt, stored in the "Logs" folder of the application -- nn is the file number). The following options are available:

- **Disable** (default)
- **With all body parts** - enabled with body parts in response and request
- **Without body parts** - enabled without body parts (body size is provided)
- **With request body** - enabled with body part in request only
- **With response body** - enabled with body part in response only

## Access Key

Defining an access key is mandatory to unlock access to the `WebAdmin` web server through a URL (access via a 4D menu command does not require an access key). When no access key is defined, no web client is allowed to connect through a URL to a web administration interface like the [Data Explorer page](#). An error page is returned in case of connection request:

An access key is similar to a password but not associated to a login.

- To define a new access key: click the **Define** button, enter the access key string in the dialog box and click **OK**. The button label becomes **Modify**.
- To modify the access key: click the **Modify** button, enter the new access key string in the dialog box and click **OK**.
- To delete the access key: click the **Modify** button, let the access key area empty and click **OK**.

## WebAdmin Headless Configuration

All [WebAdmin settings](#) are stored in the `WebAdmin.4DSettings` file. There is one default `WebAdmin.4DSettings` file per 4D and 4D Server application, so that it is possible to deploy multiple applications on the same host machine.

When running a 4D or 4D Server application headless, you can set and use the default `WebAdmin.4DSettings` file, or designate a custom `.4DSettings` file.

To set the file contents, you can use the [WebAdmin settings dialog](#) of the 4D application with interface and run it headless afterwards. The default `WebAdmin.4DSettings` file is then used.

Or, you can set a custom `.4DSettings` file (xml format) and use it instead of the default file. Several dedicated arguments are available in the [Command line interface](#) to support this feature.

The access key is not stored in clear in the `.4DSettings` file.

Example:

```
"%HOMEPATH%\Desktop\4D Server.exe" MyApp.4DLink --webadmin-access-key  
"my Fabulous AccessKey" --webadmin-auto-start true  
--webadmin-store-settings
```

## Authentication and Session

- When a web management page is accessed by entering a URL and without prior identification, an authentication is required. The user must enter the [access key](#) in an authentication dialog box. If the access key was not defined in the [WebAdmin](#) settings, no access via URL is possible.
- When a web management page is accessed directly from a 4D or 4D Server menu item (such as **Records > Data Explorer** or **Window > Data Explorer** (4D Server)), access is granted without authentication, the user is automatically authenticated.

Once the access is granted, a web [session](#) with the "WebAdmin" privilege is created on the 4D application. As long as the current session has "WebAdmin" privilege, the [WebAdmin](#) component delivers requested pages.

# Data Explorer

The Data Explorer provides a web interface to view and query data in your project datastore. Using this tool, you can easily browse among all your entities and search, order, or filter attribute values. It helps you to control data and quickly identify issues at any step of the development process.

## Access Configuration

The Data Explorer relies on the [WebAdmin](#) web server component for the configuration and authentication settings.

- **configuration:** the Data Explorer configuration reuses the [WebAdmin web server settings](#),
- **authentication:** access to the Data Explorer is granted when the [session user is authenticated](#) and has the "WebAdmin" privilege. When the Data Explorer is accessed through the **Data Explorer** menu item (see below), an automatic authentication is provided.

The Data Explorer access can be disabled using the [.setAdminProtection\(\)](#) function.

## Opening the Data Explorer

The [WebAdmin web server](#) is started automatically if necessary when the Data Explorer is clicked on.

To connect to the Data Explorer web page:

- From a 4D application (with interface):
  - To open a new 4D window with the Data Explorer page displayed in a web area, select **Data Explorer** in the **Records** menu or click on the **Data** button in the main toolbar.
  - To open the Data Explorer in an external browser tab, select **Data Explorer In Browser** in the **Records** menu press the **Alt** key (Windows)/**Option** key (macOS) and click on the **Data** button in the main toolbar.
- From 4D Server, select **Data Explorer In Browser** in the **Window** menu (the Data Explorer can only be opened on an external browser).
- Whether you use a headless 4D application or not, you can open your web browser and enter the following address:

`IPaddress:HTTPPort/dataexplorer` or `IPaddress:HTTPSPort/dataexplorer`

In this context, you will be prompted to enter the [access key](#) to open a [WebAdmin](#) session on the server:

Please enter the Access Key

Type your access key here

VALIDATE

[HTTPPort](#) and [HTTPSPort](#) values are configured in the `WebAdmin` settings.

## Using the Data Explorer

In addition to a comprehensive and customizable view of your data, the Data Explorer allows you to query and order your data.

### Requirements

The Data Explorer supports the following web browsers:

- Chrome
- Safari
- Edge
- FireFox

The minimum resolution to use the Data Explorer is 1280x720. Recommended resolution is 1920x1080.

### Basics

The Data Explorer provides an overall access to the ORDA data model with respect to the [ORDA mapping rules](#).

You can switch to the **dark mode** display theme using the selector at the bottom of the page:



The page contains several areas:

- On the left side are the **Dataclasses area** and **Attributes area**, allowing you can select the dataclasses and attributes to display. Attributes are ordered according to the underlying structure creation order. Primary key and indexed attributes have a specific icon. You can filter the list of proposed dataclass names and

A screenshot of a dark-themed interface for selecting dataclasses. At the top, it says "DataClasses". Below that is a search bar with a magnifying glass icon and a placeholder "Employee". Underneath the search bar, there is a list of items: "Employee" and "events", each preceded by a small blue square icon.

attribute names using the respective search areas.

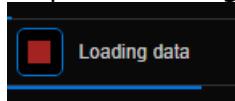
- The central part contains the **Search area** and the **Data grid** (list of entities of the selected dataclass). Each column of the grid represents a datastore attribute.

- By default, all entities are displayed. You can filter the displayed entities using the search area. Two query modes are available: [Query on attributes](#) (selected by default), and the [Advanced query with expression](#). You select the query mode by clicking on the corresponding button (the X button allows



you to reset the query area and thus stop filtering):

- The name of the selected dataclass is added as a tab above the data grid. Using these tabs, you can switch between dataclasses that have been already selected. You can remove a referenced dataclass by clicking the "remove" icon at the right of the dataclass name.
- You can reduce the number of columns by unchecking attributes in the left side. You can also switch the columns in the data grid using drag and drop. You can click on a column header to [sort entities](#) according to its values (when possible).
- If an operation requires a long time, a progress bar is displayed. You can stop the running operation at any moment by clicking on the red button:



- On the right side is the **Details area**: it displays the attribute values of the currently selected entity as well as **related data**, if any. You can browse between the entities of the dataclass by clicking the **First / Previous / Next / Last** links at the bottom of the area.
  - All attribute types are displayed, including pictures, objects (expressed in json) as well as [computed](#) and [alias](#) attributes.
  - Related data (many-to-one and one-to-many relations) can be displayed through expandable/collapsible areas:
  - Ctrl+Click** (Windows) or **Command+Click** (macOS) on a related attribute name in the right side area displays the values of the attribute in an independant, floating area:

## Updating contents

When the ORDA model or data is modified on the database side (table added, record edited or deleted, etc.), you just need to refresh the Data Explorer page in the browser (using the F5 key, for example).

## Ordering entities

You can reorder the displayed entity list according to attribute values. All types of attributes can be used for a sort, except picture and object.

- Click on a column header to order entities according to the corresponding attribute values. By default, the sort is ascending. Click twice for a descending sort. A column used to sort entities is displayed with a small icon and its name is in *italics*.



- You can sort attributes on several levels. For example, you can sort employees by city and then by salary. To do that, hold down the **Shift** key and click sequentially on each column header to include in the sort order.

## Query on attributes

In this mode, you can filter entities by entering values to find (or to exclude) in the areas above the attribute list. You can filter on one or several attributes. The entity list is automatically updated when you type in.

A screenshot of a software interface showing a table with two columns: 'ID' and 'firstname'. The table has four rows. The first row is a header with 'ID' in the first column and 'firstname' in the second. The second row contains '1' in the first column and 'Florette' in the second. The third row contains '29' in the first column and 'Flora' in the second. The fourth row contains '1 200' in the first column and 'Floria' in the second. Above the table, there is a search bar with '= ID' and a text input field containing 'Flo'. There are also three icons at the top: a magnifying glass, a comparison operator, and a delete button.

ID	firstname
1	Florette
29	Flora
1 200	Floria

If you enter several attributes, a AND is automatically applied. For example, the following filter displays entities with *firstname* attribute starting with "flo" AND *salary* attribute value > 50000:

A screenshot of a software interface showing a table with 'firstname' and 'salary' columns. The table has four rows. The first row is a header with 'firstname' in the first column and 'salary' in the second. The second row contains 'Flora' in the first column and '76 109' in the second. The third row contains 'Floria' in the first column and '58 009' in the second. The fourth row contains 'Florent' in the first column and '79 409' in the second. Above the table, there is a search bar with 'Flo' and a comparison operator '>> 50000'. There are also three icons at the top: a magnifying glass, a comparison operator, and a delete button.

firstname	salary
Flora	76 109
Floria	58 009
Florent	79 409

The **X** button allows you to remove entered attributes and thus stop filtering.

Different operators and query options are available, depending on the data type of the attribute.

You cannot filter on picture or object attributes.

## Numeric operators

With numeric, date, and time attributes, the "=" operator is selected by default. However, you can select another operator from the operator list (click on the "=" icon to display the list):

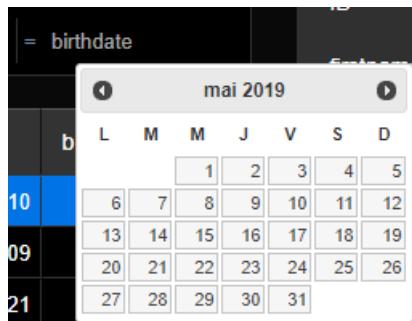
A screenshot of a software interface showing a list of numeric operators. The operators are listed vertically: '=', '<', '<=' (with 'alary' written next to it), '>', '>=' (with 'alary' written next to it), and '≠'. The '=' operator is highlighted with a blue background. Below the operators, there are two numerical values: '29 309' and '73 109'.

=	salary
<	
<=	alary
>	
>=	alary
≠	
	29 309
	73 109

## Dates

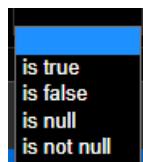
With date attributes, you can enter the date to use through a datepicker widget (click

on the date area to display the calendar):



## Booleans

When you click on a boolean attribute area, you can filter on **true/false** values but also on **null/not null** values:



- **null** indicates that the attribute value was not defined
- **not null** indicates that the attribute value is defined (thus true or false).

## Text

Text filters are not diacritic (a = A).

The filter is of the "starts with" type. For example, entering "Jim" will show "Jim" and "Jimmy" values.

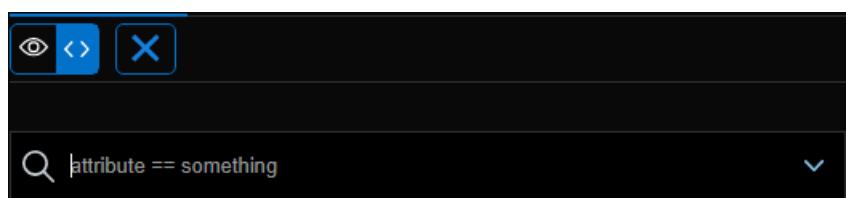
You can also use the wildcard character (@) to replace one or more starting characters. For example:

A filter with	Finds
Bel	All values beginning with "Bel"
@do	All values containing "do"
Bel@do	All values starting with "Bel" and containing "do"

If you want to create more specific queries, such as "is exactly", you may need to use the advanced queries feature.

## Advanced queries with expression

When you select this option, a query area is displayed above the entity list, allowing you to enter any expression to use to filter the contents:



You can enter advanced queries that are not available as attribute queries. For example, if you want to find entities with *firstname* attribute containing "Jim" but not "Jimmy", you can write:

```
firstname=="Jim"
```

You can use any ORDA query expression as [documented with the `query\(\)` function](#), with the following limitations or differences:

- For security, you cannot execute formulas using `eval()`.
- Placeholders cannot be used; you have to write a *queryString* with values.
- String values containing space characters must be embedded in double quotes ("").

For example, with the Employee dataclass, you can write:

```
firstname = "Marie Sophie" AND manager.lastname = "@th"
```

You can click on the  icon to display both [queryPlan](#) and [queryPath](#). In the area, you can hover over the subquery blocks to have detailed information per subquery:

Right-click in the query area to display the previous valid queries:

# Command Line Interface

You can use the macOS Terminal or the Windows console to drive your 4D applications (4D, 4D Server, merged application, and [tool4d](#)) using command lines. More particularly, this functionality allows you to:

- launch a database remotely, which can be especially useful for administering Web servers.
- run automatic tests for your applications.

## Basic information

You can execute command lines for 4D applications using the macOS Terminal or the Windows Console.

- Under macOS, you should use the `open` command.
- Under Windows, you can just pass the arguments directly.

Under macOS, you can pass the arguments directly by going to the folder where the application is found inside the package (Contents/MacOS path), which allows to address the stderr stream. For example, if the 4D package is located in the `MyFolder` folder, you must write the command line as follows:

`/MyFolder/4D.app/Contents/MacOS/4D`. However, we recommend that you use the `open` command whenever you do not need to access the stderr stream.

## Launch a 4D application

Here is a description of command lines and arguments supported to launch 4D applications.

Syntax:

```
<applicationPath> [--version] [--help] [--project] [<projectPath |  
packagePath | 4dlinkPath> [--data <dataPath>]]  
[--opening-mode interpreted | compiled] [--create-data] [--user-param  
<user string>] [--headless] [--dataless]  
[--webadmin-settings-file] [--webadmin-access-key] [--webadmin-auto-  
start] [--webadmin-store-settings]  
[--utility] [--skip-onstartup] [--startup-method <methodName string>]
```

Argument	Value	Description
<code>applicationPath</code>	Path of 4D, 4D Server, merged application, or tool4d	Launches the application. If not headless: identical to double-clicking the application; when called without structure file argument, the application is executed and the 'select database' dialog box appears.
<code>--version</code>		Displays application version and exits
<code>--help</code>		Displays help and exits. Alternate arguments: <code>-?</code> , <code>-h</code>
<code>--project</code>	<code>projectPath</code>   <code>dataPath</code>	Project file to open with the current data packagePath file. No dialog box appears.
	<code>4dlinkPath</code>	Data file to open with the designated project file. If not specified, the last

Argument	Value	Description
--opening-mode	interpreted compiled	Requests database to open in interpreted or compiled mode. No error is thrown if the requested mode is unavailable.
--create-data		Automatically creates a new data file if no valid data file is found. No dialog box appears. 4D uses the file name passed in the "--data" argument if any (generates an error if a file with the same name already exists).
--user-param	Custom user string	A string that will be available within the application through the <a href="#">Get_database_parameter</a> command (the string must not start with a "-" character, which is reserved).
--headless		<p>Launches the 4D, 4D Server or merged application without interface (headless mode). In this mode:</p> <ul style="list-style-type: none"> <li>• The Design mode is not available, database starts in Application mode</li> <li>• No toolbar, menu bar, MDI window or splash screen is displayed</li> <li>• No icon is displayed in the dock or task bar</li> <li>• The opened database is not registered in the "Recent databases" menu</li> <li>• The diagnostic log is automatically started (see <a href="#">SET DATABASE PARAMETER</a>, selector 79)</li> <li>• Every call to a dialog box is intercepted and an automatic response is provided (e.g. OK for the <a href="#">ALERT</a> command, Abort for an error dialog...). All intercepted commands(*) are logged in the diagnostic log.</li> </ul>
--dataless		<p>For maintenance needs, you can send any text to standard output streams using the <a href="#">LOG EVENT</a> command. Note that headless 4D applications can only be closed by a call to <a href="#">QUIT 4D</a> or using the OS task manager.</p> <p>Launches 4D, 4D Server, merged application or tool4d in dataless mode. Dataless mode is useful when 4D runs tasks with no need for data (project compilation for example). In this mode:</p> <ul style="list-style-type: none"> <li>• No file containing data is opened, even if specified in the command line or the <a href="#">.4DLink</a> file, or when using the <a href="#">CREATE DATA FILE</a> and <a href="#">OPEN DATA FILE</a> commands.</li> <li>• Commands that manipulate data will throw an error. For example, <a href="#">CREATE RECORD</a> throws "no table to apply the command to"</li> </ul>

Argument	Value	command to . . .	Description
		<b>Note:</b>	
		• If passed in the command line, dataless mode applies to all databases opened in 4D, as long as the application is not closed.	
		• If passed using the <code>.4DLink</code> file, dataless mode only applies to the database specified in the <code>.4DLink</code> file. For more information on <code>.4DLink</code> files, see <a href="#">Project opening shortcuts</a> .	
<code>--webadmin-settings-file</code>	File path	Path of the custom WebAdmin <code>.4DSettings</code> file for the <a href="#">WebAdmin web server</a> . Not available with <a href="#">tool4d</a> .	
<code>--webadmin-access-key</code>	String	Access key for the <a href="#">WebAdmin web server</a> . Not available with <a href="#">tool4d</a> .	
<code>--webadmin-auto-start</code>	Boolean	Status of the automatic startup for the <a href="#">WebAdmin web server</a> . Not available with <a href="#">tool4d</a> .	
<code>--webadmin-store-settings</code>		Store the access key and automatic starting parameters in the currently used settings file (i.e. the default <a href="#">WebAdmin.4DSettings</a> file or a custom file designated with the <code>--webadmin-settings-path</code> parameter). Use the <code>--webadmin-store-settings</code> argument to save these settings if necessary. Not available with <a href="#">tool4d</a> .	
<code>--utility</code>		Only available with 4D Server. Launches <a href="#">4D Server in utility mode</a> .	
<code>--skip-onstartup</code>		Launches the project without executing any "automatic" methods, including the <code>On Startup</code> and <code>On Exit</code> database methods	
<code>--startup-method</code>	Project method name (string)	Project method to execute immediately after the <code>On Startup</code> database method (if not skipped with <code>--skip-onstartup</code> ).	

(\*) Some dialogs are displayed before the database is opened, so that it's impossible to write into the [Diagnostic log file](#) (licence alert, conversion dialog, database selection, data file selection). In such case, an error message is thrown both in the stderr stream and the system event log, and then the application quits.

## Examples

The current folder of the user is reached using the "`~`" command under macOS and the "`%HOMEPATH%`" command under Windows.

Launch a 4D application stored on the desktop:

- macOS:

```
open ~/Desktop/4D.app
open "~/Desktop/4D Server.app"
```

- Windows:

```
%HOMEPATH%\Desktop\4D\4D.exe  
%HOMEPATH%\Desktop\"4D Server.exe"
```

Open a package file on macOS:

```
--args ~/Documents/myDB.4dbase
```

Open a project file:

- macOS:

```
--args ~/Documents/myProj/Project/myProj.4DProject
```

- Windows:

```
%HOMEPATH%\Documents\myProj\Project\myProj.4DProject
```

Open a project file and a data file:

- macOS:

```
--args --project ~/Documents/myProj/Project/myProj.4DProject --data  
~/Documents/data/myData.4DD
```

- Windows:

```
--project %HOMEPATH%\Documents\myProj\Project\myProj.4DProject --data  
%HOMEPATH%\Documents\data\myData.4DD
```

or:

```
/project %HOMEPATH%\Documents\myProj\Project\myProj.4DProject /data  
%HOMEPATH%\Documents\data\myData.4DD
```

Open a .4DLink file:

- macOS:

```
~/Desktop/MyDatabase.4DLink
```

- Windows:

```
%HOMEPATH%\Desktop\MyDatabase.4DLink
```

Open compiled mode and create a data file if not available:

- macOS:

```
~/Documents/myBase.4dbase --args --opening-mode compiled --create-data  
true
```

- Windows:

```
%HOMEPATH%\Documents\myBase.4dbase\myDB.4db --opening-mode compiled --  
create-data true
```

Open a project file and a data file and pass a string as a user parameter:

- macOS:

```
--args --project ~/Documents/myProj/Project/myProj.4DProject --data  
~/Documents/data/myData.4DD --user-param "Hello world"
```

- Windows:

```
--project %HOMEPATH%\Documents\myProj\Project\myProj.4DProject --data  
%HOMEPATH%\Documents\data\myData.4DD --user-param "Hello world"
```

Open without interface (headless mode):

- macOS:

```
--args --project ~/Documents/myProj/Project/myProj.4DProject --data  
~/Documents/data/myData.4DD --headless
```

- Windows:

```
--project %HOMEPATH%\Documents\myProj\Project\myProj.4DProject --data  
%HOMEPATH%\Documents\data\myData.4DD --headless
```

## tool4d

**tool4d** is a free, lightweight, stand-alone application allowing you to open a 4D project in headless mode and execute some 4D code using the CLI.

tool4d is available on Windows and macOS and is always associated to a 4D release (same version and build number). It is only provided in English localization.

tool4d is a perfect tool if you want to:

- implement a CI/CD chain for your 4D application,
- use a light 4D executable to run 4D scripts, for example to execute automatic unit tests.

### Using tool4d

You can get tool4d from the 4D [Product download page](#).

You use tool4d by executing a [command line](#) with a standard 4D project. You can use all arguments described in the above table, except `--webadmin` since this component is [disabled in tool4d](#). With tool4d, the following specific sequence is launched:

1. tool4d executes the `On Startup` database method (and all "automatic" methods such as [user method](#)), except if the `--skip-onstartup` argument is passed.
2. tool4d executes the method designated by the `--startup-method` argument, if any.
3. tool4d executes the `On Exit` database method, except if the `--skip-onstartup` argument is passed.
4. tool4d quits.

On Windows, tool4d is a console application so that the `stdout` stream is displayed in the terminal (cmd, powershell...).

### NOTES

- tool4d is always executed headless (the `headless` command line option is useless).
- The [Application type](#) command returns the value 6 ("tool4d") when called from the tool4d application.
- the [diagnostic log file](#) is prefixed with "4DDiagnosticLogTool".

### Disabled 4D features

Keep in mind that tool4d runs automatically in **headless mode** (see `--headless` in [this table](#)), and does neither give access to the 4D IDE nor any of its servers. In

particular, the following features are disabled:

- application server, Web server, SQL server,
- backup scheduler,
- ODBC and SQL pass-through,
- all components such as 4D View Pro, 4D SVG, 4D NetKit...,
- hunspell spell checker,
- japanese spellchecker (*mecab* library),
- WebAdmin,
- CEF,
- PHP,
- remote debugger (local debugger, `TRACE` command and breakpoints are ignored in headless applications).

## 4D Server in utility mode

You can launch a 4D Server instance in a utility mode (headless) by using the `--utility` CLI option. In this case, the following workflow is triggered:

1. 4D Server executes the `On Startup` database method (and all "automatic" methods such as [user method](#)), except if the `--skip-onstartup` parameter is passed.
2. 4D Server executes the method designated by the `--startup-method`, if any.
3. 4D Server executes the `On Exit` database method, except if the `--skip-onstartup` parameter is passed.
4. 4D Server quits.



INFO

Unlike `tool4d`, 4D Server in utility mode has all its features enabled. However, the application server and all other servers are not started.



SEE ALSO

See [this blog post](#) for examples of how to use `tool4d` and 4D Server in utility mode.

## TLS Protocol (HTTPS)

All 4D servers can communicate in secured mode through the TLS (Transport Layer Security) protocol:

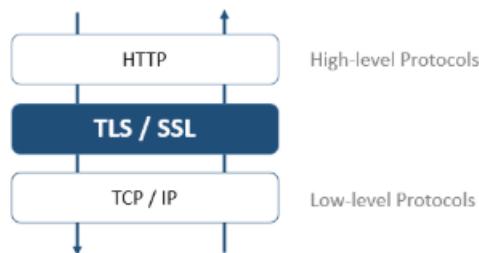
- the web server
- the application server (client-server desktop applications)
- the SQL server

## Overview

The TLS protocol (successor of SSL) has been designed to secure data exchanges between two applications —mainly between a web server and a browser. This protocol is widely used and is compatible with most web browsers.

At the network level, the security protocol is inserted between the TCP/IP layer (low level) and the HTTP high level protocol. It has been designed mainly to work with HTTP.

Network configuration using TLS:



The TLS protocol is designed to authenticate the sender and receiver and to guarantee the confidentiality and integrity of the exchanged information:

- **Authentication:** The sender and receiver identities are confirmed.
- **Confidentiality:** The sent data is encrypted so that no third person can understand the message.
- **Integrity:** The received data has not been changed, by accident or malevolently.

TLS uses a public key encryption technique based on a pair of asymmetric keys for encryption and decryption: a public key and a private key. The private key is used to encrypt data. The sender (the website) does not give it to anyone. The public key is used to decrypt the information and is sent to the receivers (web browsers) through a certificate. When using TLS with the Internet, the certificate is delivered through a certification authority, such as Verisign®. The website pays the Certificate Authority to deliver a certificate which guarantees the server authentication and contains the public key allowing to exchange data in a secured mode.

For more information on the encryption method and the public and private key issues, refer to the `ENCRYPT BLOB` command description.

## Minimum version

By default, the minimum version of the secured protocol accepted by the server is TLS 1.2. You can modify this value by using the `Min TLS version` selector with the `SET DATABASE PARAMETER` command.

You can control the level of security of your web server by defining the [minimum TLS version](#) accepted for connections.

## How to get a certificate?

A server working in secured mode means that you need a digital certificate from a certification authority. This certificate contains various information such as the site ID as well as the public key used to communicate with the server. This certificate is transmitted to the clients (e.g. Web browsers) connecting to this server. Once the certificate has been identified and accepted, the communication is made in secured mode.

Web browsers authorize only the certificates issued by a certification authority referenced in their properties.

The certification authority is chosen according to several criteria. If the certification authority is well known, the certificate will be authorized by many browsers, however the price to pay will be expensive.

To get a digital certificate:

1. Generate a private key using the `GENERATE ENCRYPTION KEYPAIR` command.  
**Warning:** For security reasons, the private key should always be kept secret. Actually, it should always remain with the server machine. For the Web server, the Key.pem file must be placed in the Project folder.
2. Use the `GENERATE CERTIFICATE REQUEST` command to issue a certificate request.
3. Send the certificate request to the chosen certificate authority. To fill in a certificate request, you might need to contact the certification authority. The certification authority checks that the information transmitted are correct. The certificate request is generated in a BLOB using the PKCS format encoded in base64 (PEM format). This principle allows you to copy and paste the keys as text and to send them via E-mail without modifying the key content. For example, you can save the BLOB containing the certificate request in a text document (using the `BLOB TO DOCUMENT` command), then open and copy and paste its content in a mail or a Web form to be sent to the certification authority.
4. Once you get your certificate, create a text file named "cert.pem" and paste the contents of the certificate into it. You can receive a certificate in different ways (usually by email or HTML form). 4D accepts all platform-related text formats for certificates (OS X, PC, Linux, etc.). However, the certificate must be in PEM format, *i.e.*, PKCS encoded in base64.

CR line-ending characters are not supported on their own; you must use CRLF or LF.

5. Place the "cert.pem" file in the [appropriate location](#).

The 4D server can now work in a secured mode. A certificate is valid between 3 months to a year.

## Installation and activation

### Installing `key.pem` and `cert.pem` files

To be able to use the TLS protocol with the server, you must install the **key.pem** (document containing the private encryption key) and **cert.pem** (document containing the certificate) at the appropriate location(s). Different locations are required depending on the server on which you want to use TLS.

Default *key.pem* and *cert.pem* files are provided with 4D. For a higher level of security, we strongly recommend that you replace these files with your own certificates.

## With the web server

To be used by the 4D web server, the **key.pem** and **cert.pem** files must be placed:

- with 4D in local mode or 4D Server, next to the [project folder](#)
- with 4D in remote mode, in the client database folder on the remote machine (for more information about the location of this folder, see the [Get 4D folder](#) command).

You must copy these files manually on the remote machine.

## With the application server (client-server desktop applications)

To be used by the 4D application server, the **key.pem** and **cert.pem** files must be placed:

- in the [Resources folder](#) of the 4D Server application
- and in the **Resources** folder on each remote 4D application (for more information about the location of this folder, see the [Get 4D folder](#) command).

## With the SQL server

To be used by the 4D SQL server, the **key.pem** and **cert.pem** files must be placed next to the [project folder](#).

## Enabling TLS

The installation of **key.pem** and **cert.pem** files makes it possible to use TLS with the 4D server. However, in order for TLS connections to be accepted by the server, you must enable them:

- With the 4D web server, you must [enable HTTPS](#). You can set the [HSTS option](#) to redirect browsers trying to connect in http mode.
- With the application server, you must select the **Encrypt Client-Server Communications** option in the "Client-server/Network options" page of the Settings dialog box.
- With the SQL server, you must select the **Enable TLS** option in the "SQL" page of the Settings dialog box.

The 4D web server also supports [HSTS option](#) to declare that browsers should only interact with it via secure HTTPS connections.

## Perfect Forward Secrecy (PFS)

[PFS](#) adds an additional layer of security to your communications. Rather than using pre-established exchange keys, PFS creates session keys cooperatively between the communicating parties using Diffie-Hellman (DH) algorithms. The joint manner in which the keys are constructed creates a "shared secret" which impedes outside parties from being able to compromise them.

When TLS is enabled on the server, PFS is automatically enabled. If the *dhparsms.pem* file (document containing the server's DH private key) does not already exist, 4D will automatically generate it with a key size of 2048. The initial generation of this file could take several minutes. The file is placed with the [\*key.pem\* and \*cert.pem\* files](#).

If you use a [\*custom cipher list\*](#) and want to enable PFS, you must verify that it contains entries with DH or ECDH (Elliptic-curve Diffie–Hellman) algorithms.

# Managing 4D Licenses

Once installed on your disk, you must activate your 4D products in order to be able to use them. Usually, the activation is automatic if you [sign in using your 4D account](#) in the Welcome Wizard.

However, in specific cases you could need to activate your licenses manually, for example if:

- your configuration does not allow the automatic activation,
- you have purchased additional licenses.

No activation is required for the following uses:

- 4D used in remote mode (connection to a 4D Server)
- 4D used in local mode with an interpreted application project with no access to the Design environment.

## First activation

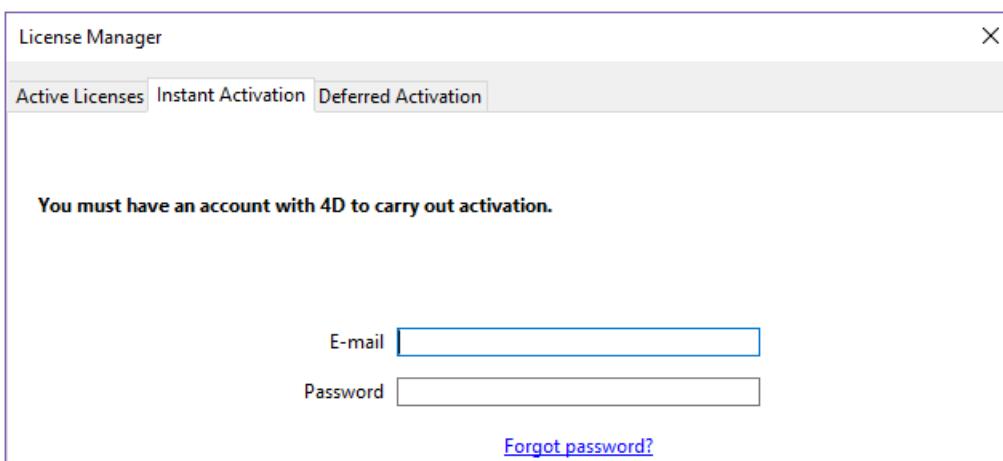
With 4D, select the **License Manager...** command from the **Help** menu of the application. With 4D Server, just launch the 4D Server application. The dialog box for choosing the [activation mode](#) appears.

4D offers three activation modes. We recommend **Instant Activation**.

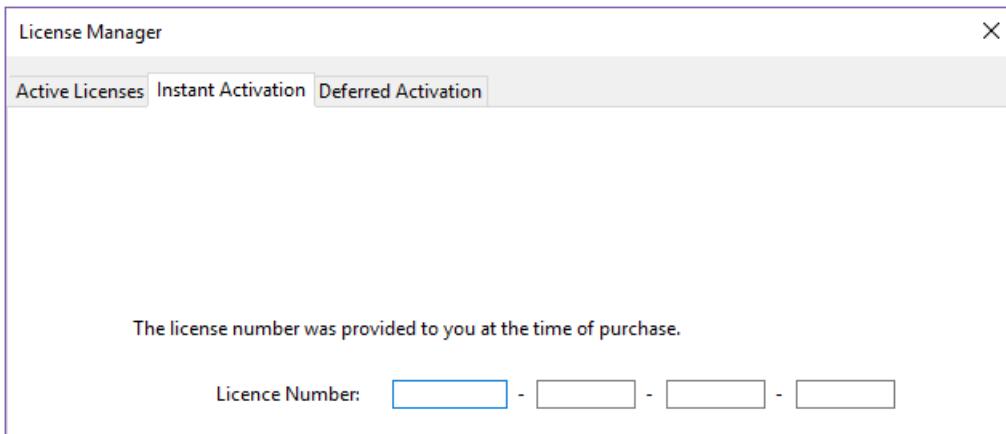
### Instant Activation

Enter your user ID (email or 4D account) as well as your password. If you do not have an existing user account, you will need to create it at the following address:

<https://account.4d.com/us/login.shtml>



Then enter the license number of the product you want to activate. This number is provided by email or by mail after a product is purchased.



## Deferred Activation

If you are unable to use [instant activation](#) because your computer does not have internet access, please proceed to deferred activation using the following steps.

1. In the License Manager window, select the **Deferred Activation** tab.
2. Enter the License Number and your e-mail address, then click **Generate file** to create the ID file (*reg.txt*).
3. Save the *reg.txt* file to a USB drive and take it to a computer that has internet access.
4. On the machine with internet access, login to <https://activation.4d.com>.
5. On the Web page, click on the **Choose File...** button and select the *reg.txt* file from steps 3 and 4; then click on the **Activate** button.
6. Download the serial file(s).



7. Save the *license4d* file(s) on a shared media and transfer them back to the 4D machine from step 1.
8. Now back on the machine with 4D, still on the **Deferred Activation** page, click **Next**; then click the **Load...** button and select a *license4d* file from the shared media from step 7.

With the license file loaded, click on **Next**.

9. Click on the **Add N°** button to add another license. Repeat these steps until all licenses from step 6 have been integrated.

Your 4D application is now activated.

## Emergency Activation

This mode can be used for a special temporary activation of 4D (5 days maximum) without connecting to the 4D Web site. This activation can only be used one time.

## Adding licenses

You can add new licenses, for example to extend the capacities of your application, at any time.

Choose the **License Manager...** command from the **Help** menu of the 4D or 4D Server application, then click on the **Refresh** button:

This button connects you to our customer database and automatically activates any new or updated licenses related to the current license (the current license is displayed in **bold** in the "Active Licenses" list). You will just be prompted for your user account and password.

- If you purchased additional expansions for a 4D Server, you do not need to enter any license number -- just click **Refresh**.
- At the first activation of a 4D Server, you just need to enter the server number and all the purchased expansions are automatically assigned.

You can use the **Refresh** button in the following contexts:

- When you have purchased an additional expansion and want to activate it,
- When you need to update an expired temporary number (Partners or evolutions).

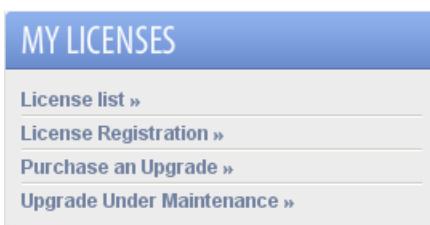
## 4D Online Store

In 4D Store, you can order, upgrade, extend, and/or manage 4D products. You can reach the store at the following address: <https://store.4d.com/us/> (you will need to select your country).

Click **Login** to sign in using your existing account or **New Account** to create a new one, then follow the on-screen instructions.

## License Management

After you log in, you can click on **License list** at the top right of the page:



Here you can manage your licenses by assigning them to projects.

Select the appropriate license from the list then click **Link to a project... >**:

You can either select an existing project or create a new one:

You can use projects to organize your licenses according to your needs:

## Troubleshooting

If the installation or activation process fails, please check the following table, which gives the most common causes of malfunctioning:

<b>Symptoms</b>	<b>Possible causes</b>	<b>Solution(s)</b>
Impossible to download product from 4D Internet site	Internet site unavailable, antivirus application, firewall	1- Try again later OR 2- Temporarily disable your antivirus application or your firewall.
Impossible to install product on disk (installation refused).	Insufficient user access rights	Open a session with access rights allowing you to install applications (administrator access)
Failure of on-line activation	Antivirus application, firewall, proxy	1- Temporarily disable your antivirus application or your firewall OR 2- Use deferred activation (not available with licenses for "R" versions)

If this information does not help you resolve your problem, please contact 4D or your local distributor.

## Contacts

For any questions about the installation or activation of your product, please contact 4D, Inc. or your local distributor.

For the US:

- Web: <https://us.4d.com/4d-technical-support>
- Telephone: 1-408-557-4600

For the UK:

- Web: <https://uk.4d.com/4d-technical-support>
- Telephone: 01625 536178

## MSC

The Maintenance and Security Center (MSC) window contains all the tools needed for verification, analysis, maintenance, backup, compacting, and encrypting of data files. The MSC window is available in all 4D applications: 4D single user, 4D Server or 4D Desktop.

**Note:** The MSC window is not available from a 4D remote connection.

There are several ways to open the MSC window. The way it is accessed also determines the way the application project is opened: in "maintenance" mode or "standard" mode. In maintenance mode, the project is not opened by 4D, only its reference is provided to the MSC. In standard mode, the project is opened by 4D.

### Display in maintenance mode

In maintenance mode, only the MSC window is displayed (the project is not opened by the 4D application). This means that projects that are too damaged to be opened in standard mode by 4D can nevertheless be accessed. Moreover, certain operations (compacting, repair, and so on) require the project to be opened in maintenance mode (see [Feature availability](#)).

You can open the MSC in maintenance mode from two locations:

- **From the standard project opening dialog box** The standard Open dialog includes the **Maintenance Security Center** option from the menu associated with the **Open** button:
- **Help/Maintenance Security Center** menu or **MSC** button in the tool bar (project not open)



When you call this function, a standard Open file dialog appears so that you can select the *.4DProject* or *.4dz* file of the to be examined. The project will not be opened by 4D.

### Display in standard mode

In standard mode, a project is open. In this mode, certain maintenance functions are not available. You have several possibilities for accessing the MSC window:

- Use the **Help/Maintenance Security Center** menu or the **MSC** button in the 4D toolbar:  
A small blue square icon containing a white shield with a star and the letters 'MSC' below it.
- Use the "msc" standard action that it is possible to associate with a menu command or a form object.
- Use the `OPEN SECURITY CENTER` language command.

### Feature availability

Certain MSC functions are not available depending on the MSC opening mode:

- Backup function is only available when the project is open (the MSC must have

been opened in standard mode).

- Data compacting, rollback, restore, repair, and encryption functions can only be used with data files that are not open (the MSC must have been opened in maintenance mode). If these functions are tried while the project is open in standard mode, a dialog warns you that it implies that the application be closed and restarted in maintenance mode.
- In encrypted databases, access to encrypted data or to the .journal file requires that a valid encryption data key be provided (see [Encrypt page](#)). Otherwise, encrypted data is not visible.

## Information Page

The Information page provides information about the 4D and system environments, as well as the database and application files. Each page can be displayed using tab controls at the top of the window.

### Program

This page indicates the name, version and location of the application as well as the active 4D folder (for more information about the active 4D folder, refer to the description of the `Get 4D folder` command in the *4D Language Reference* manual).

The central part of the window indicates the name and location of the project and data files as well as the log file (if any). The lower part of the window indicates the name of the 4D license holder, the type of license, and the name of the current 4D user.

- **Display and selection of pathnames:** On the **Program** tab, pathnames are displayed in pop-up menus containing the folder sequence as found on the disk:  
If you select a menu item (disk or folder), it is displayed in a new system window. The **Copy the path** command copies the complete pathname as text to the clipboard, using the separators of the current platform.
- **"Licenses" Folder** The **"Licenses" Folder** button displays the contents of the active Licenses folder in a new system window. All the license files installed in your 4D environment are grouped together in this folder, on your hard disk. When they are opened with a Web browser, these files display information concerning the licenses they contain and their characteristics. The location of the "Licenses" folder can vary depending on the version of your operating system. For more information about the location of this folder, refer to the `Get 4D folder` command. **\*Note:** You can also access this folder from the "Update License" dialog box (available in the Help menu).\*

### Tables

This page provides an overview of the tables in your database:

Information on this page is available in both standard and maintenance modes.

The page lists all the tables of the database (including invisible tables) as well as their characteristics:

- **ID:** Internal number of the table.
- **Tables:** Name of the table. Names of deleted tables are displayed with parenthesis (if they are still in the trash).
- **Records:** Total number of records in the table. If a record is damaged or cannot be read, *Error* is displayed instead of the number. In this case, you can consider using the verify and repair tools.
- **Fields:** Number of fields in the table. Invisible fields are counted, however, deleted fields are not counted.
- **Indexes:** Number of indexes of any kind in the table

- **Encryptable:** If checked, the **Encryptable** attribute is selected for the table at the structure level (see "Encryptable" paragraph in the Design Reference Manual).
- **Encrypted:** If checked, the records of the table are encrypted in the data file.  
\***Note:** Any inconsistency between Encryptable and Encrypted options requires that you check the encryption status of the data file in the Encrypt page of the MSC.\*
- **Address Table Size:** Size of the address table for each table. The address table is an internal table which stores one element per record created in the table. It actually links records to their physical address. For performance reasons, it is not resized when records are deleted, thus its size can be different from the current number of records in the table. If this difference is significant, a data compacting operation with the "Compact address table" option checked can be executed to optimize the address table size (see [Compact](#) page). \***Note:** Differences between address table size and record number can also result from an incident during the cache flush.\*

## Data

The **Data** page provides information about the available and used storage space in the data file.

This page cannot be accessed in maintenance mode

The information is provided in graph form:

This page does not take into account any data that may be stored outside of the data file (see "External storage").

Files that are too fragmented reduce disk, and thus, database performance. If the occupation rate is too low, 4D will indicate this by a warning icon (which is displayed on the Information button and on the tab of the corresponding file type) and specify



that compacting is necessary:



A warning icon is also displayed on the button of the [Compact](#) page:

## Activity analysis Page

The Activity analysis page allows viewing the contents of the current log file. This function is useful for parsing the use of an application or detecting the operation(s) that caused errors or malfunctions. In the case of an application in client-server mode, it allows verifying operations performed by each client machine.

It is also possible to rollback the operations carried out on the data of the database. For more information, refer to [Rollback page](#).

Every operation recorded in the log file appears as a row. The columns provide various information on the operation. You can reorganize the columns as desired by clicking on their headers.

This information allows you to identify the source and context of each operation:

- **Operation:** Sequence number of operation in the log file.
- **Action:** Type of operation performed on the data. This column can contain one of the following operations:
  - Opening of Data File: Opening of a data file.
  - Closing of Data File: Closing of an open data file.
  - Creation of a Context: Creation of a process that specifies an execution context.
  - Closing of a Context: Closing of process.
  - Addition: Creation and storage of a record.
  - Adding a BLOB: Storage of a BLOB in a BLOB field.
  - Deletion: Deletion of a record.
  - Modification: Modification of a record.
  - Start of Transaction: Transaction started.
  - Validation of Transaction: Transaction validated.
  - Cancellation of Transaction: Transaction cancelled.
  - Update context: Change in extra data (e.g. a call to `CHANGE CURRENT USER` or `SET USER ALIAS`).
- **Table:** Table to which the added/deleted/modified record or BLOB belongs.
- **Primary Key/BLOB:** contents of the primary key for each record (when the primary key consists of several fields, the values are separated by semi-colons) or sequence number of the BLOB involved in the operation.
- **Process:** Internal number of process in which the operation was carried out. This internal number corresponds to the context of the operation.
- **Size:** Size (in bytes) of data processed by the operation.
- **Date and Hour:** Date and hour when the operation was performed.
- **System User:** System name of the user that performed the operation. In client-server mode, the name of the client-side machine is displayed; in single-user mode, the session name of the user is displayed.
- **4D User:** 4D user name of the user that performed the operation. If an alias is defined for the user, the alias is displayed instead of the 4D user name.

- **Values:** Values of fields for the record in the case of addition or modification. The values are separated by ";". Only values represented in alphanumeric form are displayed.  
\***Note:** If the database is encrypted and no valid data key corresponding to the open log file has been provided, encrypted values are not displayed in this column.\*
- **Records:** Record number.

Click on **Analyze** to update the contents of the current log file of the selected application (named by default dataname.journal). The **Browse** button can be used to select and open another log file for the application. The **Export...** button can be used to export the contents of the file as text.

## Verify Page

You use this page to verify data integrity. The verification can be carried out on records and/or indexes. This page only checks the data integrity. If errors are found and repairs are needed, you will be advised to use the [Repair page](#).

## Actions

The page contains action buttons that provide direct access to the verification functions.

When the database is encrypted, verification includes validation of encrypted data consistency. If no valid data key has already been provided, a dialog requesting the passphrase or the data key is displayed.

- **Verify the records and the indexes:** Starts the total data verification procedure.
- **Verify the records only:** Starts the verification procedure for records only (indexes are not verified).
- **Verify the indexes only:** Starts the verification procedure for indexes only (records are not verified).

Verification of records and indexes can also be carried out in detail mode, table by table (see the Details section below).

## Open log file

Regardless of the verification requested, 4D generates a log file in the `Logs` folder of the application. This file lists all the verifications carried out and indicates any errors encountered, when applicable ([OK] is displayed when the verification is correct). It is created in XML format and is named: *ApplicationNameVerify\_Logyyyy-mm-dd hh-mm-ss.xml* where:

- *ApplicationName* is the name of the project file without any extension, for example "Invoices",
- *yyyy-mm-dd hh-mm-ss* is the timestamp of the file, based upon the local system time when the maintenance operation was started, for example "2019-02-11 15-20-45".

When you click on the **Open log file** button, 4D displays the most recent log file in the default browser of the machine.

## Details

The **Table list** button displays a detailed page that can be used to view and select the actual records and indexes to be checked:

Specifying the items to be verified lets you save time during the verification procedure.

The main list displays all the tables of the database. For each table, you can limit the verification to the records and/or indexes. Expand the contents of a table or the indexed fields and select/deselect the checkboxes as desired. By default, everything is selected. You can also use the **Select all**, **Deselect all**, **All records** and **All indexes** shortcut buttons.

For each row of the table, the "Action" column indicates the operations to be carried out. When the table is expanded, the "Records" and "Indexed fields" rows indicate the number of items concerned.

The "Status" column displays the verification status of each item using symbols:

 **Verification carried out with no problem**

 Verification carried out, problems encountered

 Verification partially carried out

 Verification not carried out

Click on **Verify** to begin the verification or on **Standard** to go back to the standard page.

The **Open log file** button can be used to display the log file in the default browser of the machine (see [Open log file](#) above).

The standard page will not take any modifications made on the detailed page into account: when you click on a verification button on the standard page, all the items are verified. On the other hand, the settings made on the detailed page are kept from one session to another.

## Backup Page

You can use the Backup page to view some backup parameters of the database and to launch a manual backup:

This page consists of the following three areas:

- **Backup File Destination:** displays information about the location of the application backup file. It also indicates the free/used space on the backup disk.
- **Last Backup Information:** provides the date and time of the last backup (automatic or manual) carried out on the application.
- **Contents of the backup file:** lists the files and folders included in the backup file.

The **Backup** button is used to launch a manual backup.

This page cannot be used to modify the backup parameters. To do this, you must click on the **Database properties...** button.

# Compact Page

You use this page to access the data file compacting functions.

## Why compact your files?

Compacting files meets two types of needs:

- **Reducing size and optimization of files:** Files may contain unused spaces ("holes"). In fact, when you delete records, the space that they occupied previously in the file becomes empty. 4D reuses these empty spaces whenever possible, but since data size is variable, successive deletions or modifications will inevitably generate unusable space for the program. The same goes when a large quantity of data has just been deleted: the empty spaces remain unassigned in the file. The ratio between the size of the data file and the space actually used for the data is the occupation rate of the data. A rate that is too low can lead, in addition to a waste of space, to the deterioration of database performance. Compacting can be used to reorganize and optimize storage of the data in order to remove the "holes". The "Information" area summarizes the data concerning the fragmentation of the file and suggests operations to be carried out. The [Data](#) tab on the "Information" page of the MSC indicates the fragmentation of the current data file.
- **Complete updating of data** by applying the current formatting set in the structure file. This is useful when data from the same table were stored in different formats, for example after a change in the database structure.

Compacting is only available in maintenance mode. If you attempt to carry out this operation in standard mode, a warning dialog box will inform you that the application will be closed and restarted in maintenance mode. You can compact a data file that is not opened by the application (see [Compact records and indexes](#) below).

## Standard compacting

To directly begin the compacting of the data file, click on the compacting button in the MSC window.



Since compacting involves the duplication of the original file, the button is disabled when there is not adequate space available on the disk containing the file.

This operation compacts the main file as well as any index files. 4D copies the original files and puts them in a folder named **Replaced Files (Compacting)**, which is created next to the original file. If you have carried out several compacting operations, a new folder is created each time. It will be named "Replaced Files (Compacting)\_1", "Replaced Files (Compacting)\_2", and so on. You can modify the folder where the original files are saved using the advanced mode.

When the operation is completed, the compacted files automatically replace the

original files. The application is immediately operational without any further manipulation.

When the database is encrypted, compacting includes decryption and encryption steps and thus, requires the current data encryption key. If no valid data key has already been provided, a dialog box requesting the passphrase or the data key is displayed.

**Warning:** Each compacting operation involves the duplication of the original file which increases the size of the application folder. It is important to take this into account (especially under macOS where 4D applications appear as packages) so that the size of the application does not increase excessively. Manually removing the copies of the original file inside the package can be useful in order to keep the package size down.

## Open log file

After compacting is completed, 4D generates a log file in the Logs folder of the project. This file allows you to view all the operations carried out. It is created in XML format and named: *ApplicationName\*\*\_Compact\_Log\_yyyy-mm-dd hh-mm-ss.xml*" where:

- *ApplicationName* is the name of the project file without any extension, for example "Invoices",
- *yyyy-mm-dd hh-mm-ss* is the timestamp of the file, based upon the local system time when the maintenance operation was started, for example "2019-02-11 15-20-45".

When you click on the **Open log file** button, 4D displays the most recent log file in the default browser of the machine.

## Advanced mode

The Compact page contains an **Advanced>** button, which can be used to access an options page for compacting data file.

### Compact records and indexes

The **Compact records and indexes** area displays the pathname of the current data file as well as a [...] button that can be used to specify another data file. When you click on this button, a standard Open document dialog box is displayed so that you can designate the data file to be compacted. You must select a data file that is compatible with the open structure file. Once this dialog box has been validated, the pathname of the file to be compacted is indicated in the window.

The second [...] button can be used to specify another location for the original files to be saved before the compacting operation. This option can be used more particularly when compacting voluminous files while using different disks.

### Force updating of the records

When this option is checked, 4D rewrites every record for each table during the compacting operation, according to its description in the structure. If this option is not checked, 4D just reorganizes the data storage on disk. This option is useful in the following cases:

- When field types are changed in the application structure after data were entered. For example, you may have changed a Longint field to a Real type. 4D even allows changes between two very different types (with risks of data loss), for instance a Real field can be changed to Text and vice versa. In this case, 4D does

not convert data already entered retroactively; data is converted only when records are loaded and then saved. This option forces all data to be converted.

- When an external storage option for Text, Picture or BLOB data has been changed after data were entered. This can happen when databases are converted from a version prior to v13. As is the case with the retyping described above, 4D does not convert data already entered retroactively. To do this, you can force records to be updated in order to apply the new storage mode to records that have already been entered.
- When tables or fields were deleted. In this case, compacting with updating of records recovers the space of these removed data and thus reduces file size.

All the indexes are updated when this option is selected.

### Compact address table

(option only active when preceding option is checked)

This option completely rebuilds the address table for the records during compacting. This optimizes the size of the address table and is mainly used for databases where large volumes of data were created and then deleted. In other cases, optimization is not a decisive factor.

Note that this option substantially slows compacting and invalidates any sets saved using the `SAVE SET` command. Moreover, we strongly recommend deleting saved sets in this case because their use can lead to selections of incorrect data.

#### NOTES

- Compacting takes records of tables that have been put into the Trash into account. If there are a large number of records in the Trash, this can be an additional factor that may slow down the operation.
- Using this option makes the address table, and thus the database, incompatible with the current journal file (if there is one). It will be saved automatically and a new journal file will have to be created the next time the application is launched.
- You can decide if the address table needs to be compacted by comparing the total number of records and the address table size in the [Information](#) page of the MSC.
- The [`TRUNCATE TABLE`](#) command automatically resets the address table for the specified table.

## Rollback Page

You use the Rollback page to access the rollback function among the operations carried out on the data file. It resembles an undo function applied over several levels. It is particularly useful when a record has been deleted by mistake in a database.

This function is only available when the application functions with a data log file.

If the database is encrypted and no valid data key corresponding to the open log file has been provided, encrypted values are not displayed in the **Values** column and a dialog requesting the passphrase or the data key is displayed if you click the **Rollback** button.

The contents and functioning of the list of operations are the same as for the [Activity analysis](#) window.

To perform a rollback among the operations, select the row after which all operations must be cancelled. The operation of the selected row will be the last kept. If, for example, you wish to cancel a deletion, select the operation located just before it. The deletion operation, as well as all subsequent operations, will be cancelled.

Next click on the **Rollback** button. 4D asks you to confirm the operation. If you click **OK**, the data is then restored to the exact state it was in at the moment of the selected action.

You use the menu found at the bottom of the window to select a data log file to be used when you apply the rollback function to a database restored from an archive file. In this case, you must specify the data log file corresponding to the archive.

Here is how the rollback function works: when the user clicks the **Rollback** button, 4D shuts the current database and restores the last backup of the database data. The restored database is then opened and 4D integrates the operations of the data log file up through to the selected operation. If the database has not yet been saved, 4D starts with a blank data file.

## Restore Page

You can manually restore an archive of the current application using the **Restore** page. This page provides several options that can be used to control the restoration:

4D automatic recovery systems restore applications and include data log file when necessary.

The list found in the left part of the window displays any existing backups of the application. You can also click on the **Browse...** button found just under the area in order to open any other archive file from a different location. It is then added to the list of archives.

When you select a backup in this list, the right part of the window displays the information concerning this particular backup:

- **Path:** Complete pathname of the selected backup file. Clicking the Show button opens the backup file in a system window.
- **Date and Time:** Date and time of backup.
- **Content:** Contents of the backup file. Each item in the list has a check box next to it which can be used to indicate whether or not you want to restore it. You can also use the **Check All** or **Uncheck All** buttons to set the list of items to be restored.
- **Destination folder of the restored files:** Folder where the restored files will be placed. By default, 4D restores the files in a folder named “Archivename” (no extension) that is placed next to the Project folder. To change this location, click on [...] and specify the folder where you want the restored files to be placed.

The **Restore** button launches the manual restoration of the selected element(s).

## Successive integration of several data log files

The **Integrate one or more log file(s) after restore** option allows you to integrate several data log files successively into an application. If, for example, you have 4 journal file archives (.4BL) corresponding to 4 backups, you can restore the first backup then integrate the journal (data log) archives one by one. This means that you can, for example, recover a data file even when the last backup files are missing.

When this option is checked, 4D displays the standard Open file dialog box after the restore, which can be used to select journal file to be integrated. The Open file dialog box is displayed again after each integration until it is cancelled.

## Restoring an encrypted database

Keep in mind that the data encryption key (passphrase) may have been changed through several versions of backup files (.4BK), .journal files (.4BL) and the current application. Matching encryption keys must always be provided.

When restoring a backup and integrating the current log file in a encrypted database:

- If you restore a backup using an old passphrase, this passphrase will be required at the next database startup.
- After an encryption, when opening the encrypted data file, a backup is run and a new journal file is created. Thus, it is not possible to restore a .4BK file encrypted

with one key and integrate .4BL files encrypted with another key.

The following sequence illustrates the principles of a multi-key backup/restore operation:

<b>Operation</b>	<b>Generated files</b>	<b>Comment</b>
New data file		
Add data (record # 1)		
Backup database	0000.4BL and 0001.4BK	
Add data (record # 2)		
Backup database	0001.4BL and 0002.4BK	
Add data (record # 3)		
Encrypt data file with key1	0003.4BK file (encrypted with key1)	Encryption saves original files (including journal) in folder "Replaced files (Encrypting) YYY-DD-MM HH-MM-SS". When opening the encrypted data file, a new journal is created and a backup is made to activate this journal
Add data (record #4)	0003.4BL and 0004.4BK	
Backup database	files (encrypted with key1)	We can restore 0003.4BK and integrate 0003.4BL
Add data (record # 5)	0004.4BL and 0005.4BK	
Backup database	files (encrypted with key1)	We can restore 0003.4BK and integrate 0003.4BL + 0004.4BL. We can restore 0004.4BK and integrate 0004.4BL
Add data (record # 6)		
Encrypt data file with key2	0006.4BK file (encrypted with key2)	Encryption saves original files (including journal) in folder "Replaced files (Encrypting) YYY-DD-MM HH-MM-SS". When opening the encrypted data file, a new journal is created and a backup is made to activate this journal
Add data (record # 7)	0006.4BL and 0007.4BK	
Backup database	files (encrypted)	We can restore 0006.4BK and integrate 0006.4BL

<b>Operation</b>	<b>Generated files</b>	<b>Comment</b>
Add data (record # 8)	0007.4BL and 0008.4BK	
Backup database	files (encrypted with key2)	We can restore 0006.4BK and integrate 0006.4BL + 0007.4BL. We can restore 0007.4BK and integrate 0007.4BL

When restoring a backup and integrating one or several .4BL files, the restored .4BK and .4BL files must have the same encryption key. During the integration process, if no valid encryption key is found in the 4D keychain when the .4BL file is integrated, an error is generated.

If you have stored successive data keys on the same external device, restoring a backup and integrating log files will automatically find the matching key if the device is connected.

# Repair Page

This page is used to repair the data file when it has been damaged. Generally, you will only use these functions under the supervision of 4D technical teams, when anomalies have been detected while opening the application or following a [verification](#).

**Warning:** Each repair operation involves the duplication of the original file, which increases the size of the application folder. It is important to take this into account (especially in macOS where 4D applications appear as packages) so that the size of the application does not increase excessively. Manually removing the copies of the original file inside the package can be useful to minimize the package size.

Repairing is only available in maintenance mode. If you attempt to carry out this operation in standard mode, a warning dialog will inform you that the application will be closed and restarted in maintenance mode.

When the database is encrypted, repairing data includes decryption and encryption steps and thus, requires the current data encryption key. If no valid encryption key has already been provided, a dialog requesting the passphrase or the encryption key is displayed (see Encrypt page).

## File overview

### Data file to be repaired

Pathname of the current data file. The [...] button can be used to specify another data file. When you click on this button, a standard Open document dialog is displayed so that you can designate the data file to be repaired. If you perform a [standard repair](#), you must select a data file that is compatible with the open project file. If you perform a [recover by record headers](#) repair, you can select any data file. Once this dialog has been validated, the pathname of the file to be repaired is indicated in the window.

### Original files backup folder

By default, the original data file will be duplicated before the repair operation. It will be placed in a subfolder named "Replaced files (repairing)" in the application folder. The second [...] button can be used to specify another location for the original files to be saved before repairing begins. This option can be used more particularly when repairing voluminous files while using different disks.

### Repaired files

4D creates a new blank data file at the location of the original file. The original file is moved into the folder named "¥Replaced Files (Repairing) date time" whose location is set in the "Original files backup folder" area (application folder by default). The blank file is filled with the recovered data.

### Standard repair

Standard repair should be chosen when only a few records or indexes are damaged (address tables are intact). The data is compacted and repaired. This type of repair can only be performed when the data and structure file match.

When the repair procedure is finished, the "Repair" page of the MSC is displayed. A message indicates if the repair was successful. If so, you can open the application

immediately.



The data file has been repaired.

## Recover by record headers

Use this low-level repair option only when the data file is severely damaged and after all other solutions (restoring from a backup, standard repair) have proven to be ineffective.

4D records vary in size, so it is necessary to keep the location where they are stored on disk in a specific table, named address table, in order to find them again. The program therefore accesses the address of the record via an index and the address table. If only records or indexes are damaged, the standard repair option is usually sufficient to resolve the problem. However, when the address table itself is affected, it requires a more sophisticated recovery since it will be necessary to reconstitute it. To do this, the MSC uses the marker located in the header of each record. The markers are compared to a summary of the record, including the bulk of their information, and from which it is possible to reconstruct the address table.

If you have deselected the **Records definitively deleted** option in the properties of a table in the structure, performing a recovery by header markers may cause records that were previously deleted to reappear.

Recovery by headers does not take integrity constraints into account. More specifically, after this operation you may get duplicated values with unique fields or NULL values with fields declared **Never Null**.

When you click on **Scan and repair...**, 4D performs a complete scan of the data file. When the scan is complete, the results appear in the following window:

If all the records and all the tables have been assigned, only the main area is displayed.

The "Records found in the data file" area includes two tables summarizing the information from the scan of the data file.

- The first table lists the information from the data file scan. Each row shows a group of recoverable records in the data file:
  - The **Order** column indicates the recovery order for the group of records.
  - The **Count** column indicates the number of the records in the table.
  - The **Destination table** column indicates the names of tables that were automatically assigned to the groups of identified records. The names of tables assigned automatically appear in green. Groups that were not assigned, i.e. tables that could not be associated with any records appear in red.
  - The **Recover** column lets you indicate, for each group, whether you want to recover the records. By default, this option is checked for every group with records that can be associated with a table.
- The second table lists the tables of the project file.

## Manual assigning

If several groups of records could not be assigned to tables due to a damaged address table, you can assign them manually. To do this, first select an unassigned group of records in the first table. The "Content of the records" area then displays a preview of

the contents of the first records of the group to make it easier to assign them:

Next select the table you want to assign to the group in the "Unassigned tables" table and click on the **Identify table** button. You can also assign a table using drag and drop. The group of records is then associated with the table and it will be recovered in this table. The names of tables that are assigned manually appear in black. Use the **Ignore records** button to remove the association made manually between the table and the group of records.

## Open log file

After repair is completed, 4D generates a log file in the Logs folder of the project. This file allows you to view all the operations carried out. It is created in XML format and named: *ApplicationName\*\*\_Repair\_Log\_yyyy-mm-dd hh-mm-ss.xml*" where:

- *ApplicationName* is the name of the project file without any extension, for example "Invoices",
- *yyyy-mm-dd hh-mm-ss* is the timestamp of the file, based upon the local system time when the maintenance operation was started, for example "2019-02-11 15-20-45".

When you click on the **Open log file** button, 4D displays the most recent log file in the default browser of the machine.

## Encrypt Page

You can use this page to encrypt or *decrypt* (i.e. remove encryption from) the data file, according to the **Encryptable** attribute status defined for each table in the database.

For detailed information about data encryption in 4D, please refer to the "Encrypting data" section in the *Design Reference* manual. You can also read the [A deeper look into 4D data encryption](#) blog post.

A new folder is created each time you perform an encryption/decryption operation. It is named "Replaced Files (Encrypting) yyyy-mm-dd hh-mm-ss> or "Replaced Files (Decrypting) yyyy-mm-dd hh-mm-ss".

Encryption is only available in [maintenance mode](#). If you attempt to carry out this operation in standard mode, a warning dialog will inform you that the application will be closed and restarted in maintenance mode

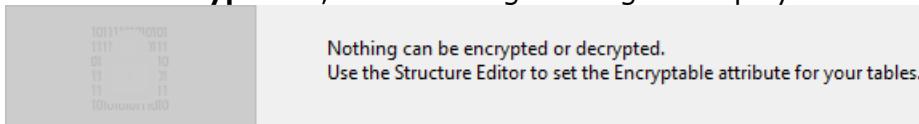
### Warning:

- Encrypting a data file is a lengthy operation. It displays a progress indicator (which could be interrupted by the user). Note also that an application encryption operation always includes a compacting step.
- Each encryption operation produces a copy of the data file, which increases the size of the application folder. It is important to take this into account (especially in macOS where 4D applications appear as packages) so that the size of the application does not increase excessively. Manually moving or removing the copies of the original file inside the package can be useful in order to minimize the package size.

## Encrypting data for the first time

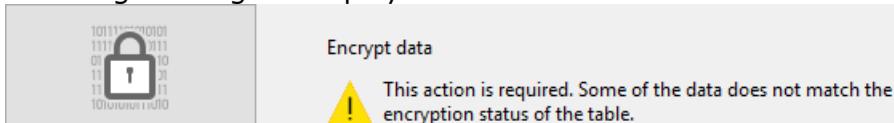
Encrypting your data for the first time using the MSC requires the following steps:

1. In the Structure editor, check the **Encryptable** attribute for each table whose data you want to encrypt. See the "Table properties" section.
2. Open the Encrypt page of the MSC. If you open the page without setting any tables as **Encryptable**, the following message is displayed in the page:



Otherwise, the

following message is displayed:

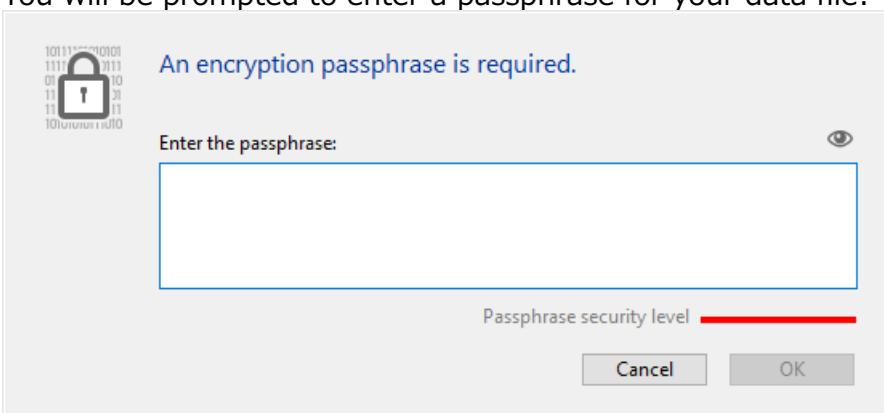


This means that the **Encryptable** status for at least one table has been modified and the data file still has not been encrypted. **Note:** The same message is displayed when the **Encryptable** status has been modified in an already encrypted data file or after the data file has been decrypted (see below).

3. Click on the Encrypt picture button.



You will be prompted to enter a passphrase for your data file:



The passphrase is used to generate the data encryption key. A passphrase is a more secure version of a password and can contain a large number of characters. For example, you could enter a passphrases such as "We all came out to Montreux" or "My 1st Great Passphrase!!" The security level indicator can help you evaluate the strength of your passphrase:  (deep green is the highest level)

#### 4. Enter to confirm your secured passphrase.

The encrypting process is then launched. If the MSC was opened in standard mode, the application is reopened in maintenance mode.

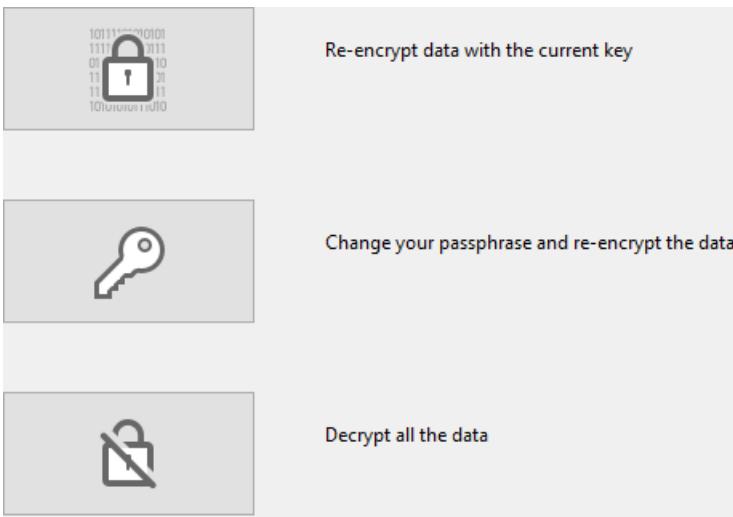
4D offers to save the encryption key (see [Saving the encryption key](#) below). You can do it at this moment or later. You can also open the encryption log file.

If the encryption process is successful, the Encrypt page displays Encryption maintenance operations buttons.

**Warning:** During the encryption operation, 4D creates a new, empty data file and fills it with data from the original data file. Records belonging to "encryptable" tables are encrypted then copied, other records are only copied (a compacting operation is also executed). If the operation is successful, the original data file is moved to a "Replaced Files (Encrypting)" folder. If you intend to deliver an encrypted data file, make sure to move/remove any unencrypted data file from the application folder beforehand.

## Encryption maintenance operations

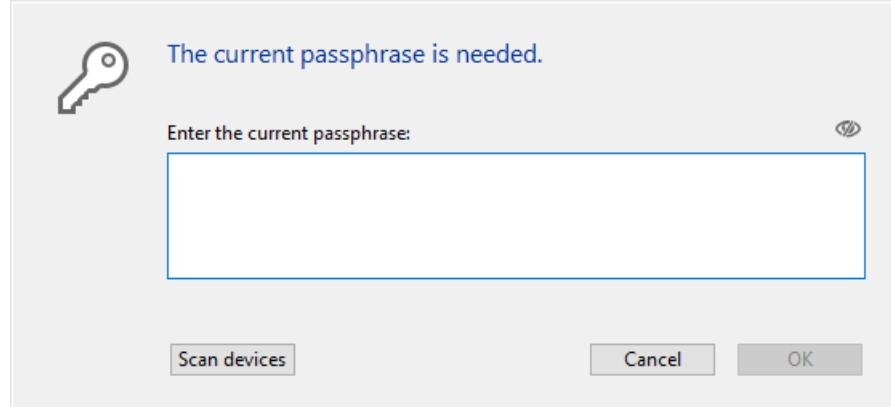
When an application is encrypted (see above), the Encrypt page provides several encryption maintenance operations, corresponding to standard scenarios.



## Providing the current data encryption key

For security reasons, all encryption maintenance operations require that the current data encryption key be provided.

- If the data encryption key is already loaded in the 4D keychain(1), it is automatically reused by 4D.
- If the data encryption key is not found, you must provide it. The following dialog



is displayed:

At this step, you have two options:

- enter the current passphrase(2) and click **OK**. OR
- connect a device such as a USB key and click the **Scan devices** button.

- (1) The 4D keychain stores all valid data encryption keys entered during the application session.
- (2) The current passphrase is the passphrase used to generate the current encryption key.

In all cases, if valid information is provided, 4D restarts in maintenance mode (if not already the case) and executes the operation.

## Re-encrypt data with the current encryption key

This operation is useful when the **Encryptable** attribute has been modified for one or more tables containing data. In this case, to prevent inconsistencies in the data file, 4D disallows any write access to the records of the tables in the application. Re-encrypting data is then necessary to restore a valid encryption status.

1. Click on **Re-encrypt data with the current encryption key**.
2. Enter the current data encryption key.

The data file is properly re-encrypted with the current key and a confirmation

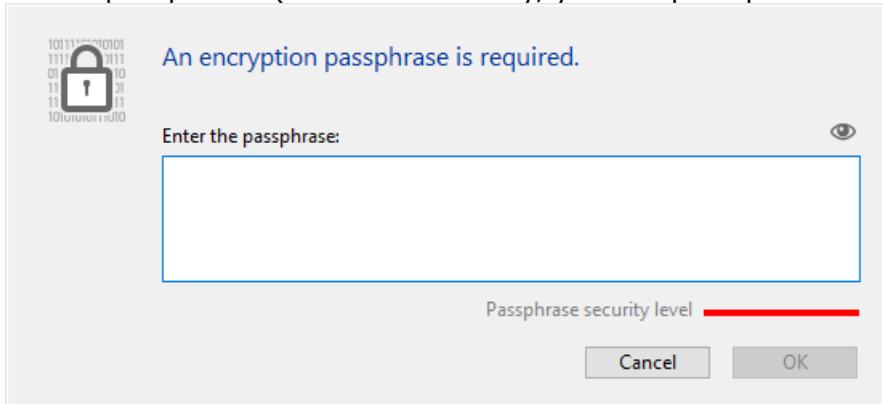
message is displayed:



## Change your passphrase and re-encrypt data

This operation is useful when you need to change the current encryption data key. For example, you may need to do so to comply with security rules (such as requiring changing the passphrase every three months).

1. Click on **Change your passphrase and re-encrypt data**.
2. Enter the current data encryption key.
3. Enter the new passphrase (for added security, you are prompted to enter it



twice):

The data

file is encrypted with the new key and the confirmation message is displayed.



## Decrypt all data

This operation removes all encryption from the data file. If you no longer want to have your data encrypted:

1. Click on **Decrypt all data**.
2. Enter the current data encryption key (see Providing the current data encryption key).

The data file is fully decrypted and a confirmation message is displayed:



Once the data file is decrypted, the encryption status of tables do not match their Encryptable attributes. To restore a matching status, you must deselect all **Encryptable** attributes at the database structure level.

## Saving the encryption key

4D allows you to save the data encryption key in a dedicated file. Storing this file on an external device such a USB key will facilitate the use of an encrypted application, since the user would only need to connect the device to provide the key before opening the application in order to access encrypted data.

You can save the encryption key each time a new passphrase has been provided:

- when the application is encrypted for the first time,
- when the application is re-encrypted with a new passphrase.

Successive encryption keys can be stored on the same device.

## **Log file**

After an encryption operation has been completed, 4D generates a file in the Logs folder of the application. It is created in XML format and named "*ApplicationName\_Encrypt\_Log\_yyyy-mm-dd hh-mm-ss.xml*" or "*ApplicationName\_Decrypt\_Log\_yyyy-mm-dd hh-mm-ss.xml*".

An Open log file button is displayed on the MSC page each time a new log file has been generated.

The log file lists all internal operations executed pertaining to the encryption/decryption process, as well as errors (if any).

## Backup and Restore

4D includes a full application backup and restore module.

This module allows backing up an application currently in use without having to exit it. Each backup can include the project folder, the data file and any additional files or folders. These parameters are first set in the Settings.

Backups can be started manually or automatically at regular intervals without any user intervention. Specific language commands, as well as specific database methods, allow integrating backup functions into a customized interface.

Applications can be restored automatically when a damaged application is opened.

Also, the integrated backup module can take advantage of the .journal file ([database log file](#)). This file keeps a record of all operations performed on the data and also ensures total security between two backups. In case of problems with an application in use, any operations missing in the data file are automatically reintegrated the next time the application is opened. You can view the journal file contents at any time.

You can also implement alternative solutions for replicating and synchronizing data in order to maintain identical versions of applications for backup purposes. These solutions can be based on the following mechanisms and technologies:

- Setting up a logical mirror with 4D Server (using the integrated backup module mechanisms)
- Synchronization using SQL
- Synchronization using HTTP (/rest/url)

For a general overview of 4D's security features, see the [4D Security guide](#).

# Backup

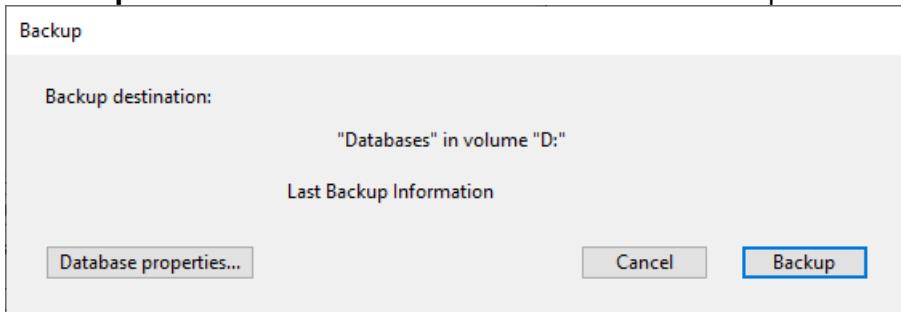
A backup can be started in three ways:

- Manually, using the **Backup...** item of the **4D File** menu or the **Backup** button of the [Maintenance and Security Center](#).
- Automatically, using the scheduler that can be set in the **Settings**,
- Programmatically, using the `BACKUP` command.

4D Server: A backup can be started manually from a remote machine using a method that calls the `BACKUP` command. The command will be executed, in all cases, on the server.

## Manual backup

1. Select the **Backup...** command in the **4D File** menu. The backup window



appears: You can see the location of the backup folder using the pop-up menu next to the "Backup destination" area. This location is set on the **Backup/Configuration** page of the Database Settings.

- You can also open the [Maintenance and Security Center](#) of 4D and display the [Backup page](#).

The **Database properties...** button causes the Backup/Configuration page of the Structure Settings to be displayed.

2. Click **Backup** to start the backup using current parameters.

## Scheduled automatic backup

Scheduled backups are started automatically. They are configured in the **Backup/Scheduler** page of the **Settings**.

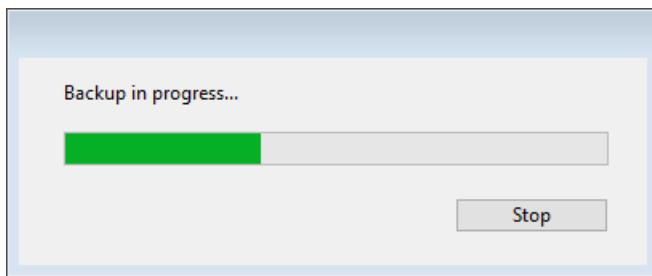
Backups are automatically performed at the times defined on this page without any type of user intervention. For more information on using this dialog box, refer to [Scheduler in backup settings](#).

## BACKUP command

When the `BACKUP` 4D language command is executed from any method, the backup starts using the current parameters as defined in the **Settings**. You can use the `On Backup Startup` and `On Backup Shutdown` database methods for handling the backup process (see the *4D Language Reference* manual).

## Managing the backup processing

Once a backup is started, 4D displays a dialog box with a thermometer indicating the progress of the backup:



This thermometer is also displayed on the [Backup page of the MSC](#) if you have used this dialog box.

The **Stop** button lets the user interrupt the backup at any time (refer to [Handling backup issues](#) below).

The status of the last backup (successful or failed) is stored in the Last Backup Information area of the [Backup page in the MSC](#) or in the [Maintenance page](#) of 4D Server. It is also recorded in the database **Backup journal.txt**.

### Accessing the application during backup

During a backup, access to the application is restricted by 4D according to the context. 4D locks any processes related to the types of files included in the backup: if only the project files are being backed up, access to the structure is not possible but access to the data will be allowed.

Conversely, if only the data file is being backed up, access to the structure is still allowed. In this case, the application access possibilities are as follows:

- With the 4D single-user version, the application is locked for both read and write; all processes are frozen. No actions can be performed.
- With 4D Server, the application is only write locked; client machines can view data. If a client machine sends an add, remove or change request to the server, a window appears asking the user to wait until the end of the backup. Once the application is saved, the window disappears and the action is performed. To cancel the request in process and not wait for the end of the backup, simply click the **Cancel operation** button. However, if the action waiting to be executed comes from a method launched prior to the backup, you should not cancel it because only operations remaining to be performed are cancelled. Also, a partially executed method can cause logical inconsistencies in the data.

When the action waiting to be executed comes from a method and the user clicks the **Cancel operation** button, 4D Server returns error -9976 (This command cannot be executed because the database backup is in progress).

### Handling backup issues

It may happen that a backup is not executed properly. There may be several causes of a failed backup: user interruption, attached file not found, destination disk problems, incomplete transaction, etc. 4D processes the incident according to the cause.

In all cases, keep in mind that the status of the last backup (successful or failed) is stored in the Last Backup Information area of the [Backup page in the MSC](#) or in the [Maintenance page](#) of 4D Server, as well as in the **Backup journal.txt**.

- **User interruption:** The **Stop** button in the progress dialog box allows users to interrupt the backup at any time. In this case, the copying of elements is stopped

and the error 1406 is generated. You can intercept this error in the `On Backup Shutdown` database method.

- **Attached file not found:** When an attached file cannot be found, 4D performs a partial backup (backup of application files and accessible attached files) and returns an error.
- **Backup impossible** (disk is full or write-protected, missing disk, disk failure, incomplete transaction, application not launched at time of scheduled automatic backup, etc.): If this is a first-time error, 4D will then make a second attempt to perform the backup. The wait between the two attempts is defined on the **Backup/Backup & Restore** page of the Settings. If the second attempt fails, a system alert dialog box is displayed and an error is generated. You can intercept this error in the `On Backup Shutdown` database method.

## Backup Journal

To make following up and verifying backups easier, the backup module writes a summary of each operation performed in a special file, which is similar to an activity journal. Like an on-board manual, all database operations (backups, restores, log file integrations) are logged in this file whether they were scheduled or performed manually. The date and time that these operations occurred are also noted in the journal.

The backup journal is named "Backup Journal[001].txt" and is placed in the "Logs" folder of the project. The backup journal can be opened with any text editor.

### Management of backup journal size

In certain backup strategies (for example, in the case where numerous attached files are being backed up), the backup journal can quickly grow to a large size. Two mechanisms can be used to control this size:

- **Automatic backup:** Before each backup, the application examines the size of the current backup journal file. If it is greater than 10 MB, the current file is archived and a new file is created with the [xxx] number incremented, for example "Backup Journal[002].txt". Once file number 999 is reached, the numbering begins at 1 again and the existing files will be replaced.
- **Possibility of reducing the amount of information recorded:** To do this, simply modify the value of the `VerboseMode` key in the *Backup.4DSettings* file of the project. By default, this key is set to True. If you change the value of this key to False, only the main information will be stored in the backup journal: date and time of start of operation and any errors encountered. The XML keys concerning backup configuration are described in the *4D XML Keys Backup* manual.

## backupHistory.json

All information regarding the latest backup and restore operations are stored in the application's **backupHistory.json** file. It logs the path of each saved file (including attachments) as well as number, date, time, duration, and status of each operation. To limit the size of the file, the number of logged operations is the same as the number of available backups ("Keep only the last X backup files") defined in the backup settings.

The **backupHistory.json** file is created in the current backup destination folder. You can get the actual path for this file using the following statement:

```
$backupHistory:=Get 4D file(Backup history file)
```

**WARNING**

Deleting or moving the **backupHistory.json** file will cause the next backup number to be reset.

The **backupHistory.json** file is formatted to be used by the 4D application. If you are looking for a human-readable report on backup operations, you might find the Backup journal more accurate.

## Backup Settings

Backup settings are defined through three pages in the [Settings dialog box](#). You can set:

- the scheduler for automatic backups
- the files to include in every backup
- the advanced features allowing to execute automatic tasks

Settings defined in this dialog box are written in the *Backup.4DSettings* file, stored in the [Settings folder](#).

## Scheduler

You can automate the backup of applications opened with 4D or 4D Server (even when no client machines are connected). This involves setting a backup frequency (in hours, days, weeks or months); for each session, 4D automatically starts a backup using the current backup settings.

If this application was not launched at the theoretical moment of the backup, the next time 4D is launched, it considers the backup as having failed and proceeds as set in the Settings (refer to [Handling backup issues](#)).

The scheduler backup settings are defined on the **Backup/Scheduler** page of the Structure Settings:

The options found on this tab let you set and configure scheduled automatic backups of the application. You can choose a standard quick configuration or you can completely customize it. Various options appear depending on the choice made in the **Automatic Backup** menu:

- **Never:** The scheduled backup feature is disabled.
- **Every Hour:** Programs an automatic backup every hour, starting with the next hour.
- **Every Day:** Programs an automatic backup every day. You can then enter the time when the backup should start.
- **Every Week:** Programs an automatic backup every week. Two additional entry areas let you indicate the day and time when the backup should start.
- **Every Month:** Programs an automatic backup every month. Two additional entry areas let you indicate the day of the month and the time when the backup should start.
- **Personalized:** Used to configure "tailormade" automatic backups. When you select this option, several additional entry areas appear:
  - **Every X hour(s):** Allows programming backups on an hourly basis. You can enter a value between 1 and 24.
  - **Every X day(s) at x:** Allows programming backups on a daily basis. For example, enter 1 if you want to perform a daily backup. When this option is checked, you must enter the time when the backup should start.
  - **Every X week(s) day at x:** Allows programming backups on a weekly basis. Enter 1 if you want to perform a weekly backup. When this option is checked, you must enter the day(s) of the week and the time when the backup should start. You can select several days of the week, if desired. For example, you can use this option to set two weekly backups: one on

Wednesday and one on Friday.

- **Every X month(s), Xth Day at x:** Allows programming backups on a monthly basis. Enter 1 if you want to perform a monthly backup. When this option is checked, you must indicate the day of the month and the time when the backup should start.

Switches back and forth from Standard time to Daylight saving time could temporarily affect the automatic scheduler and trigger the next backup with a one-hour time shift. This happens only once and subsequent backups are run at the expected scheduled time.

## Configuration

The Backup/Configuration page of the Structure Settings lets you set the backup files and their location, as well as that of the log file. These parameters are specific to each application opened by 4D or 4D Server.

**4D Server:** These parameters can only be set from the 4D Server machine.

### Content

This area allows you to set which files and/or folders to copy during the next backup.

- **Data:** Application data file. When this option is checked, the following elements are automatically backed up at the same time as the data:
  - the current log file of the application (if it exists),
  - the full `Settings` folder located [next to the data file](#) (if it exists), i.e. the *user settings for data*.
- **Structure:** Application project folders and files. In cases where projects are compiled, this option allows you to backup the .4dz file. When this option is checked, the full `Settings` folder located [at the same level as the Project folder](#), i.e. the *user settings*, is automatically backed up.
- **User Structure File (only for binary database):** *deprecated feature*
- **Attachments:** This area allows you to specify a set of files and/or folders to be backed up at the same time as the application. These files can be of any type (documents or plug-in templates, labels, reports, pictures, etc.). You can set either individual files or folders whose contents will be fully backed up. Each attached element is listed with its full access path in the “Attachments” area.
  - **Delete:** Removes the selected file from the list of attached files.
  - **Add folder...:** Displays a dialog box that allows selecting a folder to add to the backup. In the case of a restore, the folder will be recovered with its internal structure. You can select any folder or volume connected to the machine, with the exception of the folder containing the application files.
  - **Add file...:** Displays a dialog box that allows you to select a file to add to the backup.

### Backup File Destination Folder

This area lets you view and change the location where backup files as well as log backup files (where applicable) will be stored.

To view the location of the files, click in the area in order to display their pathname as a pop-up menu.

To modify the location where these files are stored, click the ... button. A selection dialog box appears, which allows you to select a folder or disk where the backups will

be placed. The "Used Space" and "Free Space" areas are updated automatically and indicate the remaining space on the disk of the selected folder.

## Log management

The **Use Log** option, when checked, indicates that the application uses a log file. Its pathname is specified below the option. When this option is checked, it is not possible to open the application without a log file.

By default, any project created with 4D uses a log file (option **Use Log File** checked in the **General Page** of the **Preferences**). The log file is named *data.journal* and is placed in the Data folder.

Activating a new log file requires the data of the application to be backed up beforehand. When you check this option, a warning message informs you that a backup is necessary. The creation of the log file is postponed and it will actually be created only after the next backup of the application.

## Backup & Restore

Modifying backup and restore options is optional. Their default values correspond to a standard use of the function.

### General settings

- **Keep only the last X backup files:** This parameter activates and configures the mechanism used to delete the oldest backup files, which avoids the risk of saturating the disk drive. This feature works as follows: Once the current backup is complete, 4D deletes the oldest archive if it is found in the same location as the archive being backed up and has the same name (you can request that the oldest archive be deleted before the backup in order to save space). If, for example, the number of sets is set to 3, the first three backups create the archives MyBase-0001, MyBase-0002, and MyBase-0003 respectively. During the fourth backup, the archive MyBase-0004 is created and MyBase-0001 is deleted. By default, the mechanism for deleting sets is enabled and 4D keeps 3 backup sets. To disable the mechanism, simply deselect the option.

This parameter concerns both application and log file backups.

- **Backup only if the data file has been modified:** When this option is checked, 4D starts scheduled backups only if data has been added, changed or deleted since the last backup. Otherwise, the scheduled backup is cancelled and put off until the next scheduled backup. No error is generated; however the backup journal notes that the backup has been postponed. This option also allows saving machine time for the backup of applications principally used for viewing purposes. Please note that enabling this option does not take any modifications made to the project files or attached files into account.

This parameter concerns both application and log file backups.

- **Delete oldest backup file before/after backup:** This option is only used if the "Keep only the last X backup files" option is checked. It specifies whether 4D should start by deleting the oldest archive before starting the backup (**before** option) or whether the deletion should take place once the backup is completed (**after** option). In order for this mechanism to work, the oldest archive must not have been renamed or moved.

- **If backup fails:** This option allows setting the mechanism used to handle failed backups (backup impossible). When a backup cannot be performed, 4D lets you carry out a new attempt.
  - **Retry at the next scheduled date and time**: This option only makes sense when working with scheduled automatic backups. It amounts to cancelling the failed backup. An error is generated.
  - **Retry after X second(s), minute(s) or hour(s)**: When this option is checked, a new backup attempt is executed after the wait period. This mechanism allows anticipating certain circumstances that may block the backup. You can set a wait period in seconds, minutes or hours using the corresponding menu. If the new attempt also fails, an error is generated and the failure is noted in the status area of the last backup and in the backup journal file.
  - **Cancel the operation after X attempts**: This parameter is used to set the maximum number of failed backup attempts. If the backup has not been carried out successfully after the maximum number of attempts set has been reached, it is cancelled and the error 1401 is generated ("The maximum number of backup attempts has been reached; automatic backup is temporarily disabled"). In this case, no new automatic backup will be attempted as long as the application has not been restarted, or a manual backup has been carried out successfully.

This parameter is useful in order to avoid a case where an extended problem (requiring human intervention) that prevented a backup from being carried out would have led to the application repeatedly attempting the backup to the detriment of its overall performance. By default, this parameter is not checked.

4D considers a backup as failed if the application was not launched at the time when the scheduled automatic backup was set to be carried out.

## Archive

These options apply to main backup files and to log backup files.

- **Segment Size (Mb)** 4D allows you to segment archives, i.e., to cut it up into smaller sizes. This behavior allows, for example, the storing of a backup on several different disks (DVDs, usb devices, etc.). During restore, 4D will automatically merge the segments. Each segment is called MyApplication[xxxx-yyyy].4BK, where xxxx is the backup number and yyyy is the segment number. For example, the three segments of the MyApplication backup are called MyApplication[0006-0001].4BK, MyApplication[0006-0002].4BK and MyApplication[0006-0003].4BK. The **Segment Size** menu is a combo box that allows you to set the size in MB for each segment of the backup. You can choose one of the preset sizes or enter a specific size between 0 and 2048. If you pass 0, no segmentation occurs (this is the equivalent of passing **None**).
- **Compression Rate** By default, 4D compresses backups to help save disk space. However, the file compression phase can noticeably slow down backups when dealing with large volumes of data. The **Compression Rate** option allows you to adjust file compression:
  - **None**: No file compression is applied. The backup is faster but the archive files are considerably larger.
  - **Fast** (default): This option is a compromise between backup speed and archive size.
  - **Compact**: The maximum compression rate is applied to archives. The archive files take up the least amount of space possible on the disk, but the backup is noticeable slowed.
- **Interlacing Rate and Redundancy Rate** 4D generates archives using specific

algorithms that are based on optimization (interlacing) and security (redundancy) mechanisms. You can set these mechanisms according to your needs. The menus for these options contain rates of **Low**, **Medium**, **High** and **None** (default).

- **\*\*Interlacing Rate\*\*:** Interlacing consists of storing data in non-adjacent sectors in order to limit risks in the case of sector damage. The higher the rate, the higher the security; however, data processing will use more memory.
- **\*\*Redundancy Rate\*\*:** Redundancy allows securing data present in a file by repeating the same information several times. The higher the redundancy rate, the better the file security; however, storage will be slower and the file size will increase accordingly.

## Automatic Restore and log integration

- **Restore last backup if database is damaged:** When this option is checked, the program automatically starts the restore of the data file of the last valid backup of the application, if an anomaly is detected (corrupted file, for example) during application launch. No intervention is required on the part of the user; however, the operation is logged in the backup journal.
- **Integrate the latest logs if the database is incomplete:** When this option is checked, the program automatically integrates the current log file if it contains operations that are not present in the data file. If there is a valid sequence of .journal files in the same repository, the program integrates beforehand all the .journal files needed from the oldest to the most current.

This situation arises, for example, if a power outage occurs when there are operations in the data cache that have not yet been written to the disk, or after an anomaly was detected when opening the data file and a restore has occurred.

### NOTE

This feature implies that the program parses all the log files in the current log file folder at startup. Therefore, for performance reasons, make sure that no useless log files are stored in the folder.

The user does not see any dialog box; the operation is completely automatic. The goal is to make use as easy as possible. The operation is logged in the backup journal.

In the case of an automatic restore, only the following elements are restored:

- .4DD file
- .4DIndx file
- .4DSyncData file
- .4DSyncHeader file
- External Data folder

If you wish to get the attached files or the project files, you must perform a [manual restore](#).

## Log file (.journal)

A continuously-used application is always recording changes, additions or deletions. Performing regular backups of data is important but does not allow (in case of incident) restoring data entered since the last backup. To respond to this need, 4D now offers a specific tool: the log file. This file allows ensuring permanent security of data.

In addition, 4D works continuously with a data cache in memory. Any changes made to the application data are stored temporarily in the cache before being written to the hard disk. This accelerates the operation of applications; in fact, accessing memory is faster than accessing the hard disk. If an incident occurs in the application before the data stored in the cache could be written to the disk, you must include the current log file in order to restore the application entirely.

Finally, 4D has functions that analyze the contents of the log file, making it possible to rollback the operations carried out on the application data. These functions are available in the MSC: refer to the [Activity analysis](#) page and the [Rollback](#) page.

## How the log file works

The log file generated by 4D contains a description of all operations performed on the data of journaled tables, which are logged sequentially. By default, all the tables are journaled, i.e. included in the log file, but you can deselect individual tables using the **Include in Log File** table property.

As such, each operation performed by a user causes two simultaneous actions: the first one in the data file (instruction is executed normally) and the second one in the log file (the description of the operation is recorded). The log file is created independently without disturbing or slowing down the work of the user. An application can only work with one log file at a time. The log file records the following action types:

- Opening and closing of the data file,
- Opening and closing of the process (contexts),
- Adding of records or BLOBs,
- Modifying of records,
- Deleting of records,
- Creating and closing of transactions.

For more information about these actions, refer to the [Activity analysis](#) page of the MSC.

4D manages the log file. It takes into account all operations that affect the data file equally, regardless of any manipulations performed by a user, a 4D method, the SQL engine, plug-ins, or from a Web browser or a mobile applicaton.

The following illustration sums up how the log file works:

The current log file is automatically saved with the current data file. This mechanism has two distinct advantages:

- Its avoids saturating the disk volume where the log file is stored. Without a backup, the log file would get bigger and bigger with use, and would eventually

use all available disk space. For each data file backup, 4D or 4D Server closes the current log file and immediately starts a new, empty file, thereby avoiding the risk of saturation. The old log file is then archived and eventually destroyed depending on the mechanism for managing the backup sets.

- It keeps log files corresponding to backups in order to be able to parse or repair an application at a later point in time. The integration of a log file can only be done in the application to which it corresponds. It is important, in order to be able to properly integrate a log file into a backup, to have backups and log files archived simultaneously.

## Creating the log file

By default, any application project created with 4D uses a log file (option set in the **General** page of the Preferences). The log file is named *data.journal* and is placed in the Data folder.

You can find out if your application uses a log file at any time: just check whether the **Use Log** option is selected on the **Backup/Configuration** page of the Settings. If you deselected this option, or if you use an application without a log file and wish to set up a backup strategy with a log file, you will have to create one.

To create a log file:

1. On the **Backup/Configuration** page of the Structure Settings, check the **Use Log** option. The program displays a standard open/new file dialog box. By default, the log file is named *data.journal*.
2. Keep the default name or rename it, and then select the file location. If you have at least two hard drives, it is recommended that you place the log file on a disk other than the one containing the application project. If the application hard drive is lost, you can still recall your log file.
3. Click **Save**. The disk and the name of the open log file are now displayed in the **Use Log** area of the dialog box. You can click on this area in order to display a pop-up menu containing the log path on the disk.
4. Validate the Settings dialog box.

In order for you to be able to create a log file directly, the data must be in one of the following situations:

- The data file is blank,
- You just performed a backup and no changes have yet been made to the data.

In all other cases, when you validate the Settings dialog box, an alert dialog box will appear to inform you that it is necessary to perform a backup. If you click **OK**, the backup begins immediately, then the log file is activated. If you click **Cancel**, the request is saved but the creation of the log file is postponed and it will actually be created only after the next backup of the application. This precaution is indispensable because, in order to restore an application after any incidents, you will need a copy of the application into which the operations recorded in the log file will be integrated.

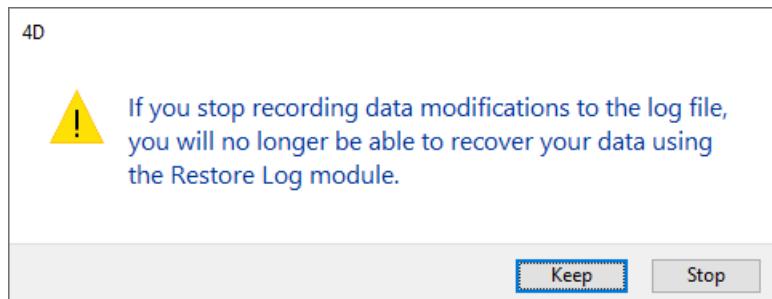
Without having to do anything else, all operations performed on the data are logged in this file and it will be used in the future when the application is opened.

You must create another log file if you create a new data file. You must set or create another log file if you open another data file that is not linked to a log file (or if the log file is missing).

## Stopping a log file

If you would like to stop logging operations to the current log file, simply deselect the **Use Log** option on the **Backup/Configuration** page of the Settings.

4D then displays an alert message to remind you that this action prevents you from taking advantage of the security that the log file provides:



If you click **Stop**, the current log file is immediately closed (the Settings dialog box does not need to be validated afterwards).

If you wish to close the current log file because it is too large, you might consider performing a data file backup, which will cause the log file to be backed up as well.

**4D Server:** The `New log file` command automatically closes the current log file and starts a new one. If for some reason the log file becomes unavailable during a working session, error 1274 is generated and 4D Server does not allow users to write data anymore. When the log file is available again, it is necessary to do a backup.

## Restore

4D allows you to restore entire sets of application data in case of any incidents, regardless of the cause of the incident. Two primary categories of incidents can occur:

- The unexpected stoppage of an application while in use. This incident can occur because of a power outage, system element failure, etc. In this case, depending on the current state of the data cache at the moment of the incident, the restore of the application can require different operations:
  - If the cache was empty, the application opens normally. Any changes made in the application were recorded. This case does not require any particular operation.
  - If the cache contains operations, the data file is intact but it requires integrating the current log file.
  - If the cache was in the process of being written, the data file is probably damaged. The last backup must be restored and the current log file must be integrated.
- The loss of application file(s). This incident can occur because of defective sectors on the disk containing the application, a virus, manipulation error, etc. The last backup must be restored and then the current log file must be integrated. To find out if an application was damaged following an incident, simply relaunch the application using 4D. The program performs a self-check and details the necessary restore operations to perform. In automatic mode, these operations are performed directly without any intervention on the part of the user. If a regular backup strategy was put into place, the 4D restore tools will allow you to recover (in most cases) the application in the exact state it was in before the incident.

4D can launch procedures automatically to recover applications following incidents. These mechanisms are managed using two options available on the **Backup/Backup & Restore** page of the Settings. For more information, refer to the [Automatic Restore](#) paragraph.

If the incident is the result of an inappropriate operation performed on the data (deletion of a record, for example), you can attempt to repair the data file using the "rollback" function in the log file. This function is available on the [Rollback](#) page of the MSC.

## Manually restoring a backup (standard dialog)

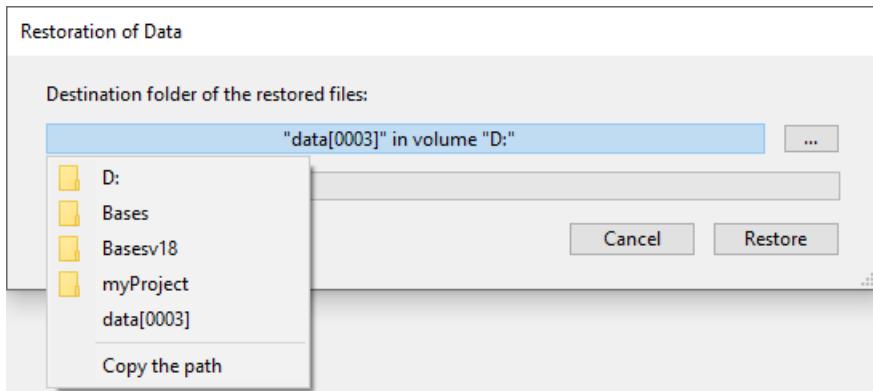
You can restore the contents of an archive generated by the backup module manually. A manual restore may be necessary, for instance, in order to restore the full contents of an archive (project files and enclosed attached files), or for the purpose of carrying out searches among the archives. The manual restore can also be performed along with the integration of the current log file.

The manual restore of backups can be carried out either via the standard Open document dialog box, or via the [Restore](#) page of the MSC. Restoring via the MSC provides more options and allows the archive contents to be previewed. On the other hand, only archives associated with the open application can be restored.

To restore an application manually via a standard dialog box:

1. Choose **Restore...** in the 4D application **File** menu. It is not mandatory that an application project be open. OR Execute the `RESTORE` command from a 4D

- method. A standard Open file dialog box appears.
2. Select a backup file (.bk) or a log backup file (.bl) to be restored and click **Open**. A dialog box appears, which allows you to specify the location where files will be restored. By default, 4D restores the files in a folder named *Archivename* (no extension) located next to the archive. You can display the path:



You can also click on the [...] button to specify a different location. 3. Click on the **Restore** button. 4D extracts all backup files from the specified location. If the current log file or a log backup file with the same number as the backup file is stored in the same folder, 4D examines its contents. If it contains operations not present in the data file, the program asks you if you want to integrate these operations. Integration is done automatically if the **Integrate last log file...** option is checked (see [Automatic Restore](#)).

4.(Optional) Click **OK** to integrate the log file into the restored application. If the restore and integration were carried out correctly, 4D displays a dialog box indicating that the operation was successful. 5. Click **OK**.

The destination folder is displayed. During the restore, 4D places all backup files in this folder, regardless of the position of the original files on the disk when the backup starts. This way your files will be easier to find.

Any content related to the data file (files and `Settings` folder) are automatically restored in a `Data` subfolder within the destination folder.

## Manually restoring a backup (MSC)

You can manually restore an archive of the current application using the [Restore page](#) of the Maintenance and Security Center (MSC).

## Manually integrating the log

If you have not checked the option for the automatic integration of the log file on the Restore page of the MSC (see [Successive integration of several log files](#)), a warning dialog box appears during the opening of the application when 4D notices that the log file contains more operations than have been carried out in the data file.

Warning



Data file does not contain the last operations.

Integrate the current log file

Open database with a new log file

Cancel

Integrate

In order for this mechanism to work, 4D must be able to access the log file in its current location.

You can choose whether or not to integrate the current log file. Not integrating the current log file allows you to avoid reproducing errors made in the data.

## **Data Collection**

To help us make our products always better, we automatically collect data regarding usage statistics on running 4D Server applications. Collected data is completely anonymous and data is transferred with no impact on the user experience.

This page explains:

- what information is collected,
- where information is stored and when it is sent to 4D,
- how to disable automatic data collection in client/server built applications.

### **Collected information**

Data is collected during the following events:

- database startup,
- database closure,
- web server startup,
- php execution,
- client connection,
- data collection sending.

Some data is also collected at regular intervals.

#### **Collected at database startup**

Data	Type	Notes
CPU	Text	Name, type, and speed of the processor
numberOfCores	Number	Total number of cores
memory	Number	Volume of memory storage (in bytes) available on the machine
system	Text	Operating system version and build number
headless	Boolean	True if the application is running in headless mode
version	Number	Version number of the 4D application
buildNumber	Number	Build number of the 4D application
license	Object	Commercial name and description of product licenses
isRosetta	Boolean	True if 4D is emulated through Rosetta on macOS, False otherwise (not emulated or on Windows).
uniqueID	Text	Unique ID of the 4D Server
id	Text (hashed string)	Unique id associated to the database ( <i>Polynomial Rolling hash of the database name</i> )
dataFileSize	Number	Data file size in bytes
indexesSize	Number	Index size in bytes
cacheSize	Number	Cache size in bytes
usingLegacyNetworkLayer	Boolean	True if legacy network layer used for the application server
usingQUICNetworkLayer	Boolean	True if the database uses the QUIC network layer
encryptedConnections	Boolean	True if client/server connections are encrypted
encrypted	Boolean	True if the data file is encrypted
compiled	Boolean	True if the application is compiled
isEngined	Boolean	True if the application is merged with 4D Volume Desktop
projectMode	Boolean	True if the application is a project
mobile	Collection	Information on mobile sessions

### Collected at web server startup and data collection sending

Data	Type	Notes
webServerObject	"started":true	: true if the web server is starting or started

### Collected at regular intervals

Data	Type	Notes
maximumNumberOfWebProcesses	Number	Maximum number of simultaneous web processes
maximumUsedPhysicalMemory	Number	Maximum use of physical memory
maximumUsedVirtualMemory	Number	Maximum use of virtual memory

### Collected at data collection sending

Data	Type	Notes
uptime	Number	Time elapsed (in seconds) since local 4D database was opened
cacheReadBytes	Object	Number of bytes read from cache
cacheMissBytes	Object	Number of bytes missed from cache
cacheReadCount	Object	Number of reads in the cache
cacheMissCount	Object	Number of reads missed in the cache
dataSegment1.diskReadBytes	Object	Number of bytes read in the data file
dataSegment1.diskWriteBytes	Object	Number of bytes written in the data file
dataSegment1.diskReadCount	Object	Number of reads in the data file
dataSegment1.diskWriteCount	Object	Number of writes in the data file
indexSegment.diskReadBytes	Number	Number of bytes read in the index file
indexSegment.diskWriteBytes	Number	Number of bytes written in the index file
indexSegment.diskReadCount	Number	Number of reads in the index file
indexSegment.diskWriteCount	Number	Number of writes in the index file

### Collected at database closure and data collection sending

Data	Type	Notes
webserverHits	Number	Number of hits on the web server during the data collection
restHits	Number	Number of hits on the REST server during the data collection
webserverBytesIn	Number	Bytes received by the web server during the data collection
webserverBytesOut	Number	Bytes sent by the web server during the data collection

### Collected every time PHP execute is called

Data	Type	Notes
phpCall	Number	Number of calls to <code>PHP execute</code>
externalPHP	Boolean	True if the client performs a call to <code>PHP execute</code> and uses its own version of php

### Collected at client connection

Data	Type	Notes
maximum4DClientConnections	Number	Maximum number of 4D Client connections to the server
connectionSystems	Collection	Client OS without the build number (in parenthesis) and number of clients using it

## Where is it stored and sent?

Collected data is written in a text file (JSON format) per database when 4D Server quits. The file is stored inside the [active 4D folder](#), i.e.:

- on Windows: `Users\[userName]\AppData\Roaming\4D Server`
- on macOS: `/Users/[userName]/Library/ApplicationSupport/4D Server`

Once a week, the file is automatically sent over the network to 4D. The file is then deleted from the active 4D folder.

If the file could not be sent for some reason, it is nevertheless deleted and

no error message is displayed on the 4D Server side.

The file is sent to the following server address: `https://dcollector.4d.com` (ip: 195.68.52.83).

## Disabling data collection in client/server built applications

You can disable the automatic data collection in [client/server built applications](#).

To disable the collection, pass the value **False** to the `ServerDataCollection` key in the `buildApp.4DSettings` file, used to build the client/server application.

## Extensions

The 4D [project architecture](#) is modular. You can provide additional functionalities to your 4D projects by installing [components](#) and [plug-ins](#). Components are made of 4D code, while plug-ins can be built using any language.

## Preinstalled 4D components

4D includes by default a set of built-in 4D components, that you can see in the **Component Methods** theme of the Explorer's Methods page.

Component Name	Description	Main Features
4D Labels	Internal component required to build label templates	
<a href="#">4D Mobile App Server</a>	Set of utility classes and functions to authenticate, manage sessions, and develop mobile applications	<code>.Action, .Authentication, .PushNotification, .WebHandler, Authentication with email confirmation</code>
<a href="#">4D NetKit</a>	Set of tools to connect to third-party APIs	<code>OAuth2Provider class, New OAuth2 provider, OAuth2ProviderObject.getToken()</code>
<a href="#">4D Progress</a>	Open one or more progress bars in the same window	<code>Progress New, Progress SET ON STOP METHOD, Progress SET PROGRESS, ...</code>
<a href="#">4D SVG</a>	Create and manipulate common svg graphic objects	<code>SVGTool_Display_viewer, multiple SVG_ methods</code>
<a href="#">4D ViewPro</a>	Spreadsheet features in your forms	See <a href="#">4D View Pro documentation</a>
<a href="#">4D Widgets</a>	Manage DatePicker, TimePicker, SearchPicker 4D widgets	<code>DatePicker calendar, DateEntry area, TimeEntry, SearchPicker SET HELP TEXT, ...</code>
<a href="#">4D WritePro Interface</a>	Manage <a href="#">4D Write Pro</a> palettes	<code>WP CreatePreview, WP PictureSettings, WP ShowTabPages, WP SwitchToolbar, WP UpdateWidget</code>

## Third-party components

You can develop and install your own 4D components. See [this page](#) for more information.

Many developers from the 4D community have shared 4D components that you can install and use in your projects.

Browse Github to have a list of public 4D components gathered with the [4d-component](#) topic.

## Plugins

Plugins do things that 4D does not natively (e.g., specific platform technology), or would be very hard to write just using 4D. As described in [this page](#), you can develop your own plug-ins.

A lot of functionnalities are covered by the existing 4D plug-ins. Browse Github to have a list of public 4D plugins gathered with the [4d-plugin](#) topic.

# 4D View Pro

## Getting Started

4D View Pro is a 4D component that includes a 4D form area and specific methods. I…

## Configuring 4D View Pro Areas

The 4D View Pro area properties can be configured using the Property list. Spreadshe…

## Formulas and Functions

Using formulas

## Method List

Warning: The commands on this page are not thread-safe.

## Classes

The following classes can be used in 4D View Pro.

## Advanced programming with Javascript

A 4D View Pro Area is a Web Area form object that uses the embedded web renderin…

## Getting Started

4D View Pro is a [4D component](#) that includes a [4D form area](#) and specific [methods](#). It allows you to embed advanced spreadsheet features in your projects.

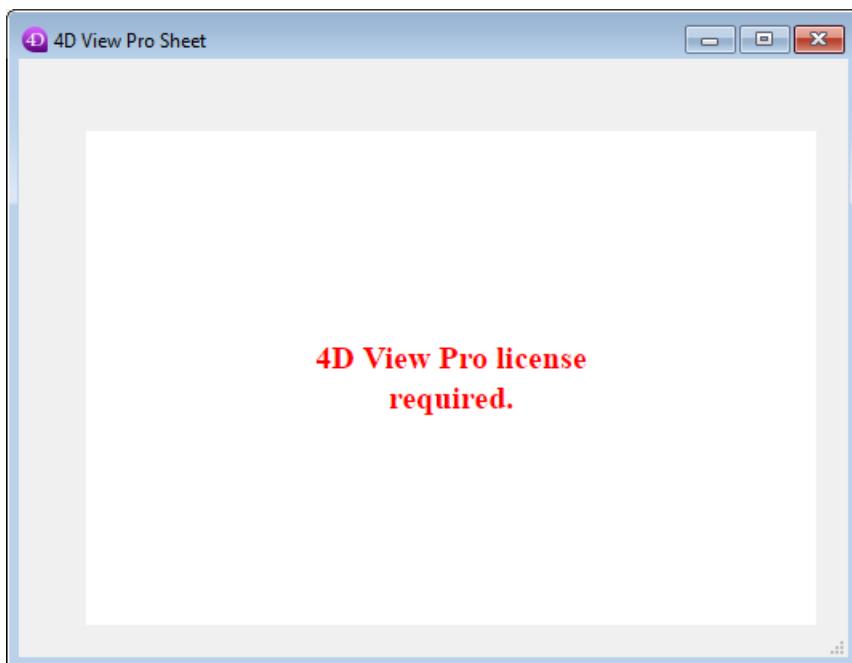
A spreadsheet is an application containing a grid of cells into which you can enter information, execute calculations, or display pictures. 4D View Pro is powered by the [SpreadJS spreadsheet solution](#) integrated in 4D.

Embedding 4D View Pro areas in your forms allows you to import and export spreadsheets documents using the 4D View Pro commands.

## Installation and activation

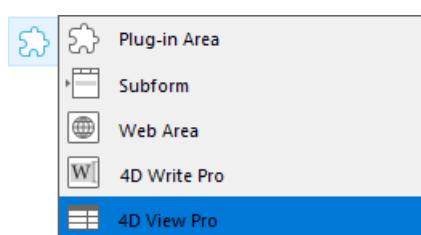
4D View Pro features are directly included in 4D, making it easy to deploy and manage. No additional installation is required.

However, 4D View Pro requires a license. You need to activate this license in your application in order to use its features. When using this component without a license, the contents of an object that requires a 4D View Pro feature are not displayed at runtime, an error message is displayed instead:



## Inserting a 4D View Pro area

4D View Pro documents are displayed and edited manually in a [4D form object](#) named 4D View Pro. To select this object, click on the last tool in the object bar:



You can also select a preconfigured 4D View Pro area in the [Object library](#).

4D View Pro areas can also be [created and used offscreen](#).

You can [configure the area](#) using the Property List and 4D View Pro methods.

## Selection, Input and Navigation Basics

Spreadsheets are composed of rows and columns. A number is associated with each row. A letter (or group of letters once the number of columns surpasses the number of letters in the alphabet) is associated with each column. The intersection of a row and a column makes a cell. Cells can be selected and their contents edited.

### Selecting cells, columns and rows

- To select a cell, simply click on it or use the direction arrows on the keyboard. Its content (or formula) is displayed within the cell.
- To select several continuous cells, drag the mouse from one end of the selection to the other. You can also click on the two ends of the selection while holding down the Shift key.
- To select all cells in the spreadsheet, click on the cell at the top left of the area:



- To select a column, click on the corresponding letter (or set of letters).
- To select a row, click on the corresponding number.
- To select a group of cells that are not continuous, hold down the **Ctrl** key (Windows) or **Command** key (Mac) and click on each cell to be selected.
- To deselect cells, simply click anywhere within the spreadsheet.

### Entering data

Double-clicking on a cell allows passing into input mode in the relevant cell. If the cell is not empty, the insertion cursor is placed after the content of the cell.



Data can be entered directly once a cell is already selected, even if the insertion cursor is not visible. The input then replaces the content of the cell.

The **Tab** key validates the cell input and selects the cell to its right. Combining the **Shift + Tab** keys validates the cell input and selects the cell to its left.

The **Carriage return** key validates the cell input and selects the cell below it. Combining the **Shift + Carriage return** keys validates the cell input and selects the cell above it.

The direction keys (arrows) allow you to move a cell in the direction indicated by the arrow.

### Using the Context Menu

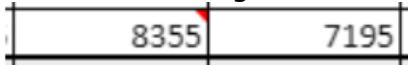
4D View Pro areas benefit from an automatic context menu that offers standard

editing features such as copy and paste, but also basic spreadsheet features:

The Copy/Cut and Paste features of the context menu only work within the spreadsheet area, they do not have access to the system pasteboard. System shortcuts such as **Ctrl+c/Ctrl+v** works however and can be used to exchange data between the area and other applications.

Depending on the clicked area, the following options are also available:

- click on a column or row header: **Insert**, **Delete**, **Hide**, or **Unhide** the contents
- click on a cell or a cell range:
  - **Filter**: allows hiding row through filters (see [Filtering rows](#) in the SpreadJS documentation).
  - **Sort**: sorts the column contents.
  - **Insert Comment**: allows user to enter a comment for an area. When a comment has been entered for an area, the top left cell of the area displays a small red triangle:



## Using 4D View Pro methods

4D View Pro methods can be used in the 4D Code Editor, just like 4D language commands.

Since 4D View Pro is a built-in 4D component, you can access its list of methods from the Explorer, in the **Component Methods** section:

For a detailed list of component methods, see [Method list](#).

### Addressing a 4D View Pro area

A 4D View Pro area handles several objects and elements.

Most of 4D View Pro methods require a *vpAreaName* parameter, which is the [\*\*4D View Pro form area name\*\*](#) (4D form object). This name is the [object name](#) property.

For example, if you want to set the total number of columns of an area named "myVpArea", you write:

```
VP SET COLUMN COUNT ("myVpArea"; 5)
```

When loading a 4D View Pro object in a form area, 4D generates the [On VP Ready](#) form event once the whole area is loaded. You must execute any 4D View Pro code handling the area in this event, otherwise an error is returned.

### Using range objects

Some 4D View Pro methods require a *rangeObj* parameter. In 4D View Pro, a range is an object that references an area in a spreadsheet. This area can be composed of one or several cells. Using 4D View Pro methods, you can create ranges and pass them to other methods to read from or write to specific locations in your document.

For example, to create a range object for the following cells:

You can use the [VP Cells](#) method:

```
var $myRange : Object  
$myRange:=VP Cells("ViewProArea";2;4;2;3) // C5 to D7
```

You can then pass `$myRange` to another 4D View Pro method to modify these cells (for example add a border to the set of cells with [VP SET BORDER](#)).

4D View Pro range objects are composed of several properties:

- area - The name of the 4D View Pro area
- ranges - A collection of range object(s). Available properties within each range object depend on the range object type. For example, a column range object will only include the `.column` and `.sheet` properties.

Property	Type	Description	Available for
area	text	4D View Pro area form object name	always available
ranges	collection	Collection of range(s)	always available
[ ].name	text	Range name	name
[ ].sheet	number	Sheet index (current sheet index by default) (counting begins at 0)	cell, cells, row, rows, column, columns, all, name
[ ].row	number	Row index (counting begins at 0)	cell, cells, row, rows
[ ].rowCount	number	Row count	cells, rows
[ ].column	number	Column index (counting begins at 0)	cell, cells, column, columns
[ ].columnCount	number	Column count	cells, columns

## Importing and exporting documents

4D View Pro supports the import and export of several document formats:

- .4vp
- .xlsx
- .txt and .csv
- .pdf (for export only)

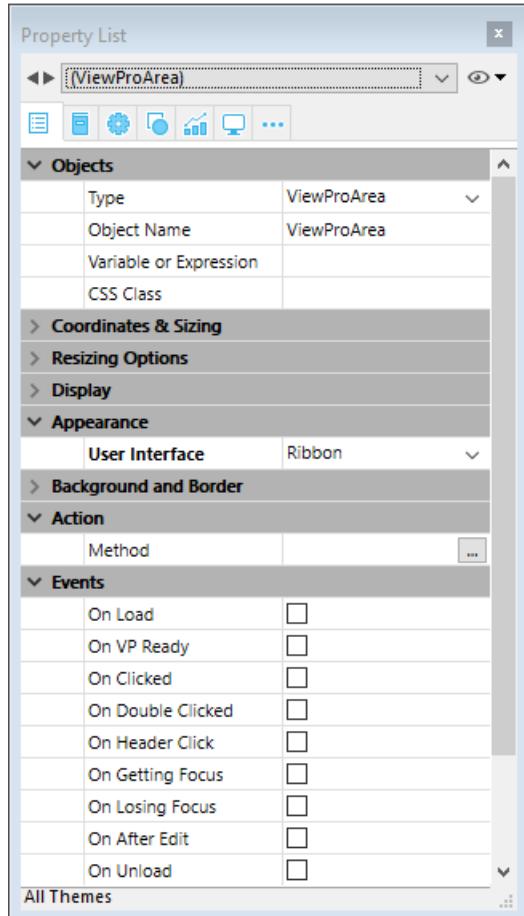
For more details, check out the description of [VP IMPORT DOCUMENT](#) and [VP EXPORT DOCUMENT](#).

# Configuring 4D View Pro Areas

The 4D View Pro area properties can be configured using the Property list. Spreadsheet properties are available through the language.

## Form area properties

Using the area's property list, you can set [4D View Pro object properties](#) such as **Object Name**, [Variable or Expression](#), **Appearance**, **Action**, and **Events**.



## Selecting a user interface

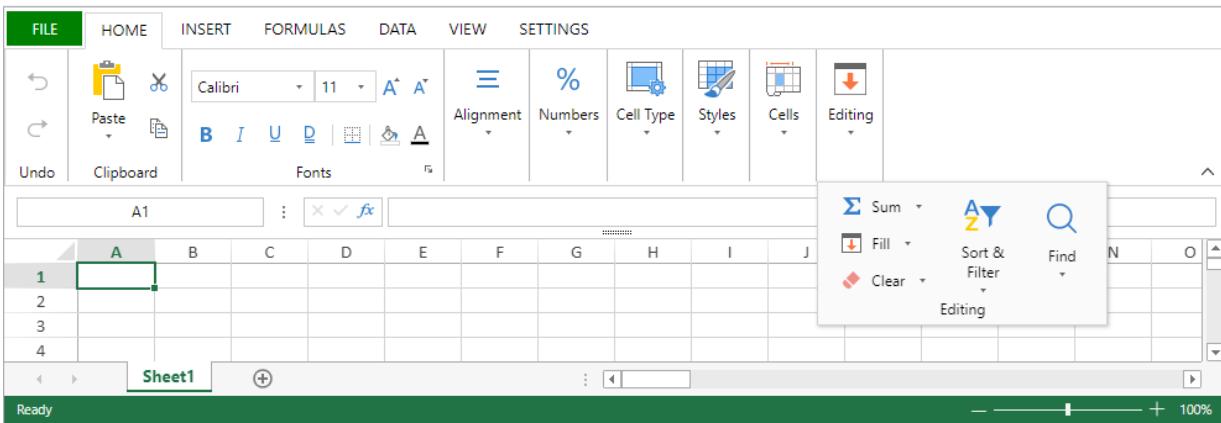
You can select the interface to use with your 4D View Pro form areas in the **Property List**, under **Appearance**:



You can also use the `userInterface` and `withFormulaBar` (only with the "toolbar" interface) JSON properties.

Interfaces allow for basic modifications and data manipulation. User-defined modifications are saved in the 4D View Pro object when the user saves the document.

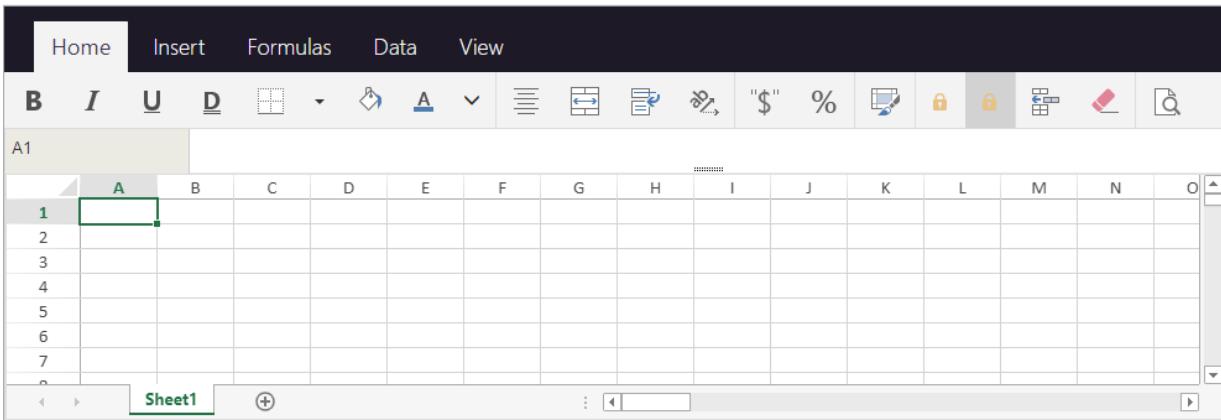
### Ribbon



## Toolbar

Enabling the Toolbar interface displays the [Show Formula Bar](#) option. When selected, the formula bar is visible below the Toolbar interface.

With visible formula bar:



## Features

Both the Ribbon and the Toolbar interfaces group related features into tabs:

Tab	Actions	Ribbon Interface	Toolbar Interface
File	File manipulation	X	
Home	Text appearance	X	X
Insert	Add items	X	X
Formulas	Formula calculation and library	X	X
Data	Data manipulation	X	X
View	Visual presentation	X	X
Settings	Sheet presentation reference	X	

## Form Events

The following form events are available in the Property List for 4D View Pro areas.

Some of the events are standard form events (available to all active objects) and some are specific 4D View Pro form events. Some standard form events provide extended information in the object returned by the [FORM\\_Event](#) command when they are generated for 4D View Pro areas. The following table shows which events are standard and which are specific or provide additional information to 4D View Pro areas:

## Standard 4D events Specific and extended 4D View Pro events

<a href="#">On Load</a>	<a href="#">On VP Ready</a>
<a href="#">On Getting Focus</a>	<a href="#">On Clicked</a>
<a href="#">On Losing Focus</a>	<a href="#">On Double Clicked</a>
<a href="#">On Unload</a>	<a href="#">On Header Click</a>
	<a href="#">On After Edit</a>
	<a href="#">On Selection Change</a>
	<a href="#">On Column Resize</a>
	<a href="#">On Row Resize</a>
	<a href="#">On VP Range Changed</a>

## Sheet Options

The 4D View Pro sheet options object allows you to control various options of your 4D View Pro areas. This object is handled by the following commands:

- [VP SET SHEET OPTIONS](#)
- [VP Get sheet options](#)

### Sheet appearance

Property	Type	Description	
allowCellOverflow	boolean	Specifies whether data can overflow into adjacent empty cells.	
sheetTabColor	string	A color string used to represent the sheet tab color, such as "red", "#FFFF00", "rgb(255,0,0)", "Accent 5", and so on.	
frozenlineColor	string	A color string used to represent the frozen line color, such as "red", "#FFFF00", "rgb(255,0,0)", "Accent 5", and so on.	
clipBoardOptions	longint	The clipboard option. Available values: <code>vk clipboard paste options all</code> , <code>vk clipboard paste options formatting</code> , <code>vk clipboard paste options formulas</code> , <code>vk clipboard paste options formulas and formatting</code> , <code>vk clipboard paste options values</code> , <code>vk clipboard paste options values and formatting</code>	
gridline	object	The grid line's options.	
	color	A color string used to represent the grid line color, such as "red", "#FFFF00", "rgb(255,0,0)", "Accent 5", and so on.	
	showVerticalGridline	boolean	Specifies whether to show the vertical grid line.
	showHorizontalGridline	boolean	Specifies whether to show the horizontal grid line.
rowHeaderVisible	boolean	Specifies whether the row header is visible.	
		Specifies whether the column	

Property	Type	Description
rowHeaderAutoText	longint	Specifies whether the row header displays letters or numbers or is blank. Available values: <code>vk_header_auto_text_blank</code> , <code>vk_header_auto_text_letters</code> , <code>vk_header_auto_text_numbers</code>
colHeaderAutoText	longint	Specifies whether the column header displays letters or numbers or is blank. Available values: <code>vk_header_auto_text_blank</code> , <code>vk_header_auto_text_letters</code> , <code>vk_header_auto_text_numbers</code>
selectionBackColor	string	The selection's background color for the sheet. (preferred RGBA format)
selectionBorderColor	string	The selection's border color for the sheet.
sheetAreaOffset	object	The sheetAreaOffset's options.
left	longint	The offset left of sheet from host.
top	longint	The offset top of sheet from host.

All properties are optional.

## Sheet protection

To lock the whole sheet, you only need to set the `isProtected` property to **true**. You can then unlock cells individually by setting the [locked](#) cell style property.

<b>Property</b>	<b>Type</b>	<b>Description</b>
isProtected	boolean	Specifies whether cells on this sheet that are marked as protected cannot be edited.
protectionOptions	object	A value that indicates the elements that you want users to be able to change. If null : the protectionOptions parameter is reset.
allowSelectLockedCells	boolean	Specifies whether the user can select locked cells, optional. True by default.
allowSelectUnlockedCells	boolean	Specifies whether the user can select unlocked cells, optional. True by default.
allowSort	boolean	Specifies whether the user can sort ranges, optional. False by default.
allowFilter	boolean	Specifies whether the user can filter ranges, optional. False by default.
allowEditObjects	boolean	Specifies whether the user can edit floating objects, optional. False by default.
allowResizeRows	boolean	Specifies whether the user can resize rows, optional. False by default.
allowResizeColumns	boolean	Specifies whether the user can resize columns, optional. False by default.
allowDragInsertRows	boolean	Specifies whether the user can perform the drag operation to insert rows, optional. False by default.
allowDragInsertColumns	boolean	Specifies whether the user can perform the drag operation to insert columns, optional. False by default.
allowInsertRows	boolean	Specifies whether the user can insert rows, optional. False by default.
allowInsertColumns	boolean	Specifies whether the user can insert columns, optional. False by default.
allowDeleteRows	boolean	Specifies whether the user can delete rows, optional. False by default.
allowDeleteColumns	boolean	Specifies whether the user can delete columns, optional. False by default.

All properties are optional.

## Cell Format

Defining a format pattern ensures that the content of your 4D View Pro documents is displayed the way you intended. Formats can be set using the selected 4D View Pro

[interface](#), or using the [VP SET VALUE](#) or [VP SET NUM VALUE](#) methods.

4D View Pro has built-in formats for numbers, dates, times, and text, but you can also create your own patterns to format the contents of cells using special characters and codes.

For example, when using the [VP SET VALUE](#) or [VP SET NUM VALUE](#) methods to enter amounts in an invoice, you may want the currency symbols (\$, €, ¥, etc.) to be aligned regardless of the space required by the number (i.e., whether the amount is \$5.00 or \$5,000.00). You could use formatting characters and specify the pattern (\$\* #,##0.00) which would display amounts as shown:

\$ 4,180.00
\$ 15.00
\$ 200.00
\$ 15,600.00
\$ 1,672.00

Note that when creating your own format patterns, only the display of the data is modified. The value of the data remains unchanged.

## Number and text formats

Number formats apply to all number types (e.g., positive, negative, and zeros).

Character	Description	Example
0	Placeholder that displays zeros.	#.00 will display 1.1 as 1.10
.	Displays a decimal point	0.00 will display 1999 as 1999.00
,	Displays the thousands separator in a number. Thousands are separated by commas if the format contains a comma enclosed by number signs "#" or by zeros. A comma following a digit placeholder scales the number by 1,000.	,0 will display 12200000 as 12,200,000
_	Skips the width of the next character.	Usually used in combination with parentheses to add left and right indents, _( and _) respectively.
@	Formatter for text. Applies the format to all text in the cell	"[Red]@" applies the red font color for text values.
*	Repeats the next character to fill the column width.	0- will include enough dashes after a number to fill the cell, whereas 0 before any format will include leading zeros.
" "	Displays the text within the quotes without interpreting it.	"8%" will display as: 8%
%	Displays numbers as a percentage of 100.	8% will be displayed as .08
#	Digit placeholder that does not display extra zeros. If a number has more digits to the right of the decimal than there are placeholders, the number is rounded up.	#.# will display 1.54 as 1.5
?	Digit placeholder that leaves space for extra zeros, but does not display them. Typically used to align numbers by decimal point.	\$?? displays a maximum of 2 decimals and causes dollar signs to line up for varying amounts.
¥	Displays the character following it. When used with numbers, displays them as fractions. When used with text, date or time codes, displayed "as-is".	#.00¥? will display 123 as 123.00?
/	Creates conditional formats.	#/# will display .75 as 3/4
[ ]	Scientific notation format.	[>100][GREEN]#,##0;[<=100][YELLOW]#,##0;[BLUE]#,##0 #E+# - will display 1,500,500 as 2E+6
[color]	Formats the text or number in the color specified	[Green]####.[##][Red]-###[.###[

## Example

```
//Set the cell value as $125,571.35
VP SET VALUE(VP Cell("ViewProArea";3;2);New
object("value";125571.35;"format";"_($* #,##0.00_)")
```

## Date and time formats

4D View Pro provides the following constants for ISO 8601 date and time patterns:

Constant	Value	Comment
vk pattern full date time	"fullDateTimePattern"	ISO 8601 format for the full date and time in current localization.USA default pattern: "dddd, dd MMMM yyyy HH:mm:ss"
vk pattern long date	"longDatePattern"	ISO 8601 format for the full date in current localization.USA default pattern: "dddd, dd MMMM yyyy"
vk pattern long time	"longTimePattern"	ISO 8601 format for the time in current localization.USA default pattern: "HH:mm:ss"
vk pattern month day	"monthDayPattern"	ISO 8601 format for the month and day in current localization.USA default pattern: "MMMM dd"
vk pattern short date	"shortDatePattern"	Abbreviated ISO 8601 format for the date in current localization.USA default pattern: "MM/dd/yyyy"
vk pattern short time	"shortTimePattern"	Abbreviated ISO 8601 format for the time in current localization.USA default pattern: "HH:mm"
vk pattern sortable date time	"sortableDateTimePattern"	ISO 8601 format for the date and time in current localization which can be sorted.USA default pattern: "yyyy-MM-dd'T'HH:mm:ss"
vk pattern universal sortable date time	"universalSortableDateTimePattern"	ISO 8601 format for the date and time in current localization using UTC which can be sorted.USA default pattern: "yyyy-MM-dd'T'HH:mm:ss'Z"
vk pattern year month	"yearMonthPattern"	ISO 8601 format for the month and year in current localization.USA default pattern: "yyyy MMMM"

## Example

```
//Set the cell value as specific date and time
VP SET VALUE(VP Cell("ViewProArea";3;9);New object("value";!2024-12-18!);"time";?14:30:10?;"format";vk pattern full date time))
```

## Custom date and time formats

To create your own date and time patterns, in your current localization, you can use combinations of the following codes:

<b>Code (not case-sensitive)</b>	<b>Description</b>	<b>Example</b>
Date		(January 1, 2019)
m	Month number without leading zero	1
mm	Month number with leading zero	01
mmm	Month name, short	Jan
mmmm	Month name, long	January
d	Day number without leading zero	1
dd	Day number with leading zero	01
ddd	Day of week, short	Tue
dddd	Day of week, long	Tuesday
yy	Year, short	19
yyyy	Year, long	2019
Time		(2:03:05 PM)
h	Hour without leading zero. 0-23	2
hh	Hour with leading zero. 00-23	02
m	Minutes without leading zero. 0-59	3
mm	Minutes with leading zero. 00-59	03
s	Seconds without leading zero. 0-59	5
ss	Seconds with leading zero. 00-59	05
[h]	Elapsed time in hours	14 (can exceed 24)
[mm]	Elapsed time in minutes	843
[ss]	Elapsed time in seconds	50585
AM/PM	Periods of day. 24 hour fomat used if omitted.	PM

The code 'm' is interpreted depending on its position in the pattern. If it's immediately after 'h' or 'hh' or immediately before 's' or 'ss', it will be interpreted as minutes, otherwise it will be interpreted as months.

## Additional symbols

In addition to the special characters and codes described in the previous sections, there are additional characters and symbols that can be used in your format patterns. These additional characters and symbols do not require a \$ or "" and do not impact the interpretation of the format pattern. They appear "as-is" within the pattern.

Character	Description	Example
+	Plus and minus signs	### + ### =
( )	Left and right parenthesis	###,###
:	Colon	(-###.###)
^	Caret	hh:mm:ss
'	Apostrophe	#¥^#
{ }	Curly brackets	'####/#
< >	Less-than and greater than signs	{###,###,###}
=	Equal sign	# # > # #
/	Forward slash. When used with numbers, displays them as fractions.	#+#=##
!	Exclamation point	mm/dd/yyyy
&	Ampersand	\$###.00!
~	Tilde	"Hello" & "Welcome"
Space character		~# #
€	Euro	€###.00
£	British Pound	£###.00
¥	Japanese Yen	¥###.00
\$	Dollar sign	\$###.00
¢	Cent sign	.00¢

## Print Attributes

4D View Pro print attributes allow you to control all aspects of printing 4D View Pro areas. These attributes are handled by the following commands:

- [VP SET PRINT INFO](#)
- [VP Get print info](#)

## Columns / Rows

Column and row attributes are used to specify the beginning, end, and repetition of columns and rows.

Property	Type	Description
columnEnd	longint	The last column to print in a cell range. Default value = -1 (all columns)
columnStart	longint	The first column to print in a cell range. Default value = -1 (all columns)
repeatColumnEnd	longint	The last column of a range of columns to print on the left of each page. Default value = -1 (all columns)
repeatColumnStart	longint	The first column of a range of columns to print on the left of each page. Default value = -1 (all columns)
repeatRowEnd	longint	The last row of a range of rows to print on the top of each page. Default value = -1 (all rows)
repeatRowStart	longint	The first row of a range of rows to print at the top of each page. Default value = -1 (all rows)
rowEnd	longint	The last row to print in a cell range. Default value = -1 (all rows)
rowStart	longint	The first row to print in a cell range. Default value = -1 (all rows)

## Headers / Footers

Header and footer attributes are used to specify text or images in the left, right, and center header/footer sections.

Property	Type	Description
footerCenter	text	The text and format of the center footer on printed pages.
footerCenterImage	picture   text*	The image for the center section of the footer.
footerLeft	text	The text and format of the left footer on printed pages.
footerLeftImage	picture   text*	The image for the left section of the footer.
footerRight	text	The text and format of the right footer on printed pages.
footerRightImage	picture   text*	The image for the right section of the footer.
headerCenter	text	The text and format of the center header on printed pages.
headerCenterImage	picture   text*	The image for the center section of the header.
headerLeft	text	The text and format of the left header on printed pages.
headerLeftImage	picture   text*	The image for the left section of the header.
headerRight	text	The text and format of the right header on printed pages.
headerRightImage	picture   text*	The image for the right section of the header.

\* If using text type, pass the filepath (absolute or relative) of the image. If you pass a relative path, the file should be located next to the database structure file. In Windows, the file extension must be indicated. No matter the type used to set an image, the image itself (not a reference) is stored in the 4D View Pro area and is returned by [VP Get print info](#).

## Special Characters

The following special characters allow the automatic addition or formatting of information in the header and footer when the 4D View Pro area is printed.

Character	Description	Example	Result
&	Escape character	(see examples below)	
P	Current page	printInfo.headerLeft:="This is page &P."	This is page 5.
N	Page count	printInfo.headerLeft:="There are &N pages."	There are 10 pages.
D	Current date (yyyy/mm/dd format)	printInfo.headerLeft:="It is &D."	It is 2015/6/19.
T	Current time	printInfo.headerLeft:="It is &T."	It is 16:30:36.
G	Image	printInfo.headerLeftImage:=smiley printInfo.headerLeft:="&G"	
S	Strikethrough	printInfo.headerLeft:="&SThis is text."	<del>This is text.</del>
U	Underline	printInfo.headerLeft:="&UThis is text."	This is text. (Underlined)
B	Bold	printInfo.headerLeft:="&BThis is text."	<b>This is text.</b>
I	Italic	printInfo.headerLeft:="&IThis is text."	<i>This is text.</i>
"	Font prefix	printInfo.headerLeft:="&¥"Lucida Console¥"&14This is text."	<b>This is text.</b>
K	Text Color prefix	printInfo.headerLeft:="&KFF0000This is text."	This is text (in red).
F	Workbook name	printInfo.headerLeft:="&F"	2019 Monthly Revenue Forecasts
A	Spreadsheet name	printInfo.headerLeft:="&A"	June 2019 revenue forecast

## Margins

Margin attributes are used to specify the 4D View Pro area margins for printing. Expressed in hundreds of an inch.

Property	Type	Description
margin	object	The print margins
top	longint	Top margin, in hundredths of an inch. Default value = 75
bottom	longint	Bottom margin, in hundredths of an inch. Default value = 75
left	longint	Left margin, in hundredths of an inch. Default value = 70
right	longint	Right margin, in hundredths of an inch. Default value = 70
header	longint	Header offset, in hundredths of an inch. Default value = 30
footer	longint	Footer offset, in hundredths of an inch. Default value = 30

## Orientation

Orientation attributes are used to specify the direction the printed page layout.

This attribute defines rendering information only.

Property	Type	Description
orientation	longint	Page orientation. Available values: <code>vk_print_page_orientation_landscape</code> , <code>vk_print_page_orientation_portrait</code> (default)

## Page

Page attributes are used to specify general document print settings.

Property	Type	Description
		Printing in black and white only.
blackAndWhite	boolean	Default value = false <b>Note:</b> PDFs are not affected by this attribute. Colors in PDFs remain.
centering	longint	How the contents are centered on the printed page. Available values: <code>vk print centering both</code> , <code>vk print centering horizontal</code> , <code>vk print centering none</code> (default), <code>vk print centering vertical</code>
firstPageNumber	longint	The page number to print on the first page. Default value = 1
pageOrder	longint	The order pages are printed. Available values: <code>vk print page order auto</code> (default), <code>vk print page order down then over</code> , <code>vk print page order over then down</code> .
pageRange	text	The range of pages for printing
qualityFactor	longint	The quality factor for printing (1 - 8). The higher the quality factor, the better the printing quality, however printing performance may be affected.
		Default value = 2
useMax	boolean	Only columns and rows with data are printed. Default value = true
zoomFactor	real	The amount to enlarge or reduce the printed page. Default value = 1

## Paper Size

Paper size attributes are used to specify the dimensions or model of paper to use for printing. There are two ways to define paper size:

- Custom size - height and width attributes
- Standard size - kind attribute

Property	Type	Description
paperSize	object	Paper dimensions (height, width) or specific format (kind) for printing.  height longint Height of the paper, in hundredths of an inch. width longint Width of the paper, in hundredths of an inch.
kind	text	Name of standard paper size (e.g., A2, A4, legal, etc.) returned by <code>Get Print Option</code> . Default value = "letter"

- If the paper size is specified using the `height` and `width` properties, [VP Get print info](#) returns a paper size with `custom` as value for `kind`.
- If you set the paper size using the `kind` property, you can use either:
  - one of the formats in the [SpreadJS format list](#)

- one of the formats returned by the [PRINT OPTION VALUES](#) command. In that case, [VP Get print info](#) returns the corresponding format with the height and width.

## Scale

Scale attributes are used to specify printing optimization and adjustments.

<b>Property</b>	<b>Type</b>	<b>Description</b>
bestFitColumns	boolean	Column width is adjusted to fit the largest text width for printing. Default value = "false"
bestFitRows	boolean	Row height is adjusted to fit the tallest text height for printing. Default value = "false"
fitPagesTall	longint	The number of vertical pages (portrait orientation) to check when optimizing printing. Default value = -1
fitPagesWide	longint	The number of horizontal pages (landscape orientation) to check when optimizing printing. Default value = -1

## Show / Hide

Show / Hide attributes are used to specify the visibility (printing) of 4D View Pro area elements.

<b>Property</b>	<b>Type</b>	<b>Description</b>
showBorder	boolean	Prints the outline border. Default value = "true" Column header print settings. Available values: <code>vk print visibility hide</code> , <code>vk print visibility inherit (default)</code> , <code>vk print visibility show</code> , <code>vk print visibility show once</code>
showColumnHeader	longint	
showGridLine	boolean	Prints the gridlines. Default value = "false" Row headers print settings. Available values: <code>vk print visibility hide</code> , <code>vk print visibility inherit (default)</code> , <code>vk print visibility show</code> , <code>vk print visibility show once</code>
showRowHeader	longint	

## Watermark

Watermark attributes are used to superimpose text or an image onto the 4D View Pro area.

<b>Property</b>	<b>Type</b>	<b>Description</b>
watermark	collection	Collection of watermark settings. Default value: <code>undefined</code>
[ ].height	longint	The height of the watermark text / image.
[ ].picture	picture	The watermark text / image.
[ ].imageSrc	text*	The page(s) where the watermark is printed. For all pages: "all". For specific pages: page numbers or page ranges separated by commas. Ex.: "1,3,5-12"
[ ].width	longint	The width of the watermark text / image.
[ ].x	longint	The horizontal coordinate of the top left point of the watermark text / image.
[ ].y	longint	The vertical coordinate of the top left point of the watermark text / image.

\* If using text type, pass the filepath (absolute or relative) of the image. If you pass a

relative path, the file should be located next to the database structure file. In Windows, the file extension must be indicated. No matter the type used to set an image, the image itself (not a reference) is stored in the 4D View Pro area and is returned by [VP Get print info](#).

## Style Objects

4D View Pro style objects and style sheets allow you to control the graphical aspects and the look of your 4D View Pro documents.

### Style objects & Style sheets

Style objects contain the style settings. They can be used either in a style sheet or on their own. Style objects can also be used in addition to a style sheet so that different settings can be specified for individual cell ranges without affecting the rest of the document. You can use style objects directly with the [VP SET CELL STYLE](#) and [VP SET DEFAULT STYLE](#) commands. You can also use style objects when defining custom table themes using the [VP SET TABLE THEME](#) or [VP CREATE TABLE](#) commands.

A **style sheet** groups together a combination of properties in a style object to specify the look of all of the cells in your 4D View Pro documents. Style sheets saved with the document can be used to set the properties for a single sheet, multiple sheets, or an entire workbook. When created, a 4D View Pro style sheet is given a name which is saved within the style sheet in the "name" property. This allows a style sheet to be easily used and, if thoughtfully selected, can facilitate its identification and purpose (e.g., Letterhead\_internal, Letterhead\_external).

Style sheets are created with the [VP ADD STYLESHEET](#) command and applied with the the [VP SET DEFAULT STYLE](#) or [VP SET CELL STYLE](#) commands. You can remove a style sheet with the [VP REMOVE STYLESHEET](#) command.

The [VP Get stylesheet](#) command can be used to return the style object of a single style sheet or you can use the [VP Get stylesheets](#) command to retrieve a collection of style objects for multiple style sheets.

### Style object properties

Example:

```
$style:=New object
$style.hAlign:=vk horizontal align left
$style.font:="12pt papyrus"
$style.backColor:="#E6E6FA" //light purple color

VP SET DEFAULT STYLE ("myDoc";$style)
```

### Background & Foreground

<b>Property</b>	<b>Type</b>	<b>Description</b>	<b>Possible values</b>
backColor	text	Defines the color of the background.	CSS color "#rrggbb" syntax (preferred syntax), CSS color "rgb(r,g,b)" syntax (alternate syntax), CSS color name (alternate syntax)
backgroundImage	picture, text	Specifies a background image.	Can be specified directly or via the image path (full path or file name only). If the file name only is used, the file must be located next to the database structure file. No matter how set (picture or text), a picture is saved with the document. This could impact the size of a document if the image is large. Note for Windows: File extension must be included.
backgroundImageLayout longint		Defines the layout for the background image.	<code>vk image layout center, vk image layout none, vk image layout stretch, vk image layout zoom</code>
foreColor	text	Defines the color of the foreground.	CSS color "#rrggbb" syntax (preferred syntax), CSS color "rgb(r,g,b)" syntax (alternate syntax), CSS color name (alternate syntax)

## Borders

<b>Property</b>	<b>Type</b>	<b>Description</b>	<b>Possible values</b>
borderBottom, borderLeft, borderRight, borderTop, diagonalDown, diagonalUp		Defines the object corresponding border line	
	color text	Defines the color of the border. Default = black.	CSS color "#rrggbb" syntax (preferred syntax), CSS color "rgb(r,g,b)" syntax (alternate syntax), CSS color name (alternate syntax)
	style longint	Defines the style of the border. Default = empty. Cannot be null or undefined.	<code>vk line style dash dot, vk line style dash dot dot, vk line style dashed, vk line style dotted, vk line style double, vk line style empty, vk line style hair, vk line style medium, vk line style medium dash dot, vk line style medium dash dot dot, vk line style medium dashed, vk line style slanted dash dot, vk line style thick</code>

## Fonts and text

<b>Property</b>	<b>Type</b>	<b>Description</b>	<b>Possible values</b>
		Specifies the font characteristics in CSS font shorthand ("font-style font-variant font-weight font-	A CSS font shorthand.

Property	Type	Description	Possible values
font	text	<p>size/line-height font-family"). Example: "14pt Century Gothic". The font-size and font-family values are mandatory. If one of the other values is missing, their default values are used.</p> <p>Note: If a font name contains a space, the name must be within quotes.</p>	4D Possible values commands to handle font characteristics as objects: <a href="#">VP Font to object</a> and <a href="#">VP Object to font</a>
formatter	text	Pattern for value/time property.	Number/text/date/time formats, special characters. See <a href="#">Cell Format</a> .
isVerticalText	boolean	Specifies text direction.	True = vertical text, False = horizontal text.
labelOptions	object	Defines cell label options (watermark options).	<pre>vk label alignment top left, vk label alignment bottom left, vk label alignment top center, vk label alignment bottom center, vk label alignment top right, vk label alignment bottom right  vk label visibility auto, vk label visibility hidden, vk label visibility visible</pre>
alignment	longint	Specifies the position of the cell label. Optional property.	<pre>css color "#rrggb" syntax (preferred syntax), CSS color "rgb(r,g,b)" syntax (alternate syntax), CSS color name (alternate syntax)</pre>
visibility	longint	Specifies the visibility of the cell label. Optional property.	
foreColor	text	Defines the color of the foreground. Optional property.	
font	text	Specifies the font characteristics with CSS font shorthand ("font-style font-variant font-weight font-size/line-height font-family"). The font-size and font-family values are mandatory.	<pre>vk text decoration double underline, vk text decoration line through, vk text decoration</pre>
textDecoration	longint	Specifies the decoration	

Property	Type	Description	Possible values
		added to text decoration	none, <code>vk_text</code>
		overline, <code>vk_text</code>	
		decoration	
		underline	
textIndent	longint	Defines the unit of text indentation. 1 = 8 pixels	
textOrientation	longint	Defines the rotation angle of the text in a cell. Number between -90 and 90	
watermark	text	Defines the watermark (cell label) content	
wordWrap	boolean	Specifies if text should be wrapped.	True = wrapped text, False = unwrapped text

## Layout

Property	Type	Description	Possible values
cellPadding	text	Defines the cell padding	
hAlign	longint	Defines the horizontal alignment of cell contents.	<code>vk_horizontal_align_center</code> , <code>vk_horizontal_align_general</code> , <code>vk_horizontal_align_left</code> , <code>vk_horizontal_align_right</code>
locked	boolean	Specifies cell protection status. Note, this is only available if <a href="#">sheet protection</a> is enabled.	True = locked, False = unlocked.
shrinkToFit	boolean	Specifies if the contents of the cell should be reduced.	True = reduced content, False = no reduction.
tabStop	boolean	Specifies if the focus to the cell can be set using the Tab key.	True = Tab key sets focus, False = Tab key does not set focus.
vAlign	longint	Specifies the vertical alignment of cell contents.	<code>vk_vertical_align_bottom</code> , <code>vk_vertical_align_center</code> , <code>vk_vertical_align_top</code>

## Style information

Property	Type	Description
name	text	Defines the name of the style
parentName	text	Specifies the style that the current style is based on. Values from the parent style will be applied, then any values from the current style are applied. Changes made in the current style will not be reflected in the parent style. Only available when using a style sheet.

## 4D View Pro Object

The 4D View Pro [object](#) stores the whole spreadsheet contents. It is automatically handled by 4D View Pro. You can set or get this object using the [VP IMPORT FROM OBJECT](#) or [VP Export to object](#) methods.

It contains the following properties:

<b>Property</b>	<b>Value type</b>	<b>Description</b>
version	Longint	Internal component version
dateCreation	Timestamp	Creation date
dateModified	Timestamp	Last modification date
meta	Object	Free contents, reserved for the 4D developer
spreadJS	Object	Reserved for the 4D View Pro component

## 4D View Pro Form Object Variable

The 4D View Pro form object variable is the [object](#) variable associated to the 4D View Pro form area. It manages information used by the 4D View Pro object.

The 4D View Pro form object variable is for information purposes only (i.e., debugging). Under no circumstances should it be modified.

It contains the following properties:

<b>Property</b>	<b>Value type</b>	<b>Description</b>
ViewPro.area	Text	4D View Pro area name
ViewPro.callbacks	Object	Stores temporary information necessary for commands requiring callbacks such as importing and exporting.
ViewPro.commandBuffersCollection	Object	Stores sequentially the commands called by the method and executes them as a batch (rather than individually) upon exiting the method, or if a command returns a value or the <a href="#">VP_FLUSH_COMMANDS</a> is called. This mechanism increases performance by reducing the number of requests sent.
ViewPro.events	Object	<a href="#">Event</a> list.
ViewPro.formulaBar	Boolean	Indicates whether or not the formula bar is displayed. Available only for the "toolbar" interface.
ViewPro.initiated	Boolean	Indicates whether or not the 4D View Pro area has been initialized (see <a href="#">On VP Ready</a> event).
ViewPro.interface	Text	Specifies the type of user interface:"ribbon", "toolbar", "none".

# Formulas and Functions

## Using formulas

A spreadsheet formula is an expression that calculates the value of a cell.

### Entering formulas

To enter a formula in a 4D View Pro area:

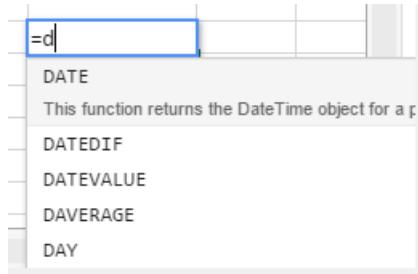
1. Select the cell into which you will enter the formula or function.
2. Enter = (the equal sign).
3. Type the formula and hit the **Enter** key.

When writing a formula, you can use different shortcuts:

- click on a cell to enter its reference in the formula:

100
200
=B2+B3

- type the first letter of a function to enter. A pop-up menu listing the available functions and references appears, allowing you to select the desired elements:



You can also create named formulas that can be called via their name. To do so, enter these formulas using the [VP ADD FORMULA NAME](#) command.

## Operators and Operands

All formulas have operands and operators:

- **Operators:** see [Values and operators](#) below.
- **Operands** include several categories:
  - [values](#) (5 data types are supported)
  - [references to other cells](#) (relative, absolute, mixed or by name)
  - [standard spreadsheet functions](#)
  - [4D functions](#) based upon 4D formulas and providing access to 4D variables, fields, methods, commands, or expressions.

## Values and operators

4D View Pro supports five types of data. For each data type, specific literal values and operators are supported.

Data types	Values	Operators
<u>Number</u>	1.2	+ (addition) - (subtraction)
	1.2 E3	* (multiplication) / (division)
	1.2E-3	^ (exponent, the number of times to multiply a number by itself)
	10.3x	% (percentage -- divide the number before the operator by one hundred)
<u>Date</u>	10/24/2017	+ (date + number of days -> date) + (date + time -> date + time of day) - (date - number of days -> date) - (date - date -> number of days between the two)
		Duration operators: + (addition)
		- (subtraction)
		(duration number -> duration) / (duration / number -> duration)
<u>String</u>	'Sophie' or "Sophie"	& (concatenation)
<u>Boolean</u>	TRUE or FALSE	-

## Comparison operators

The following operators can be used with two operands of the same type:

Operator	Comparison
=	equal to
<>	different than
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

## Operator precedence

List of operators from most to least important:

Operator	Description
()	Parenthesis (for grouping)
-	Negate
+	Plus
%	Percent
^	Exponent
* and /	Multiply and divide
+ and -	Add and Subtract
&	Concatenate
= > < >= <= <>	Compare

## Cell references

Formulas often refer to other cells by cell addresses. You can copy these formulas into

other cells. For example, the following formula, entered in cell C8, adds the values in the two cells above it and displays the result.

= C6 + C7

This formula refers to cells C6 and C7. That is, 4D View Pro is instructed to refer to these other cells for values to use in the formula.

When you copy or move these formulas to new locations, each cell address in that formula will either change or stay the same, depending on how it is typed.

- A reference that changes is called a **relative reference**, and refers to a cell by how far left/right and up/down it is from the cell with the formula.
- A reference that always points to a particular cell is called an **absolute reference**.
- You can also create a mixed reference which always points to a fixed row or column.

## Reference Notation

If you use only cell coordinates, for example, `C5`, 4D View Pro interprets the reference as relative. You may make the reference an absolute reference by putting a dollar sign in front of the letter and the number, as in `$C$5`.

You can mix absolute and relative references by inserting a dollar sign in front of the letter or the number alone, for example, `$C5` or `C$5`. A mixed reference allows you to specify either the row or the column as absolute, while allowing the other portion of the address to refer relatively.

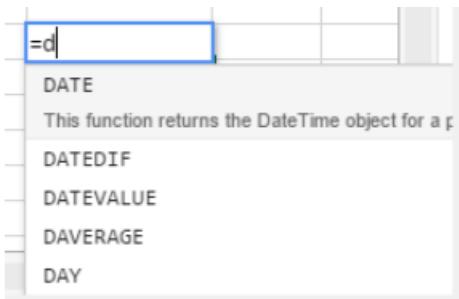
A convenient, fast and accurate way to specify an absolute reference is to name the cell and use that name in place of the cell address. A reference to a named cell is always absolute. You can create or modify named cells or named cell ranges using the [VP\\_ADD\\_RANGE\\_NAME](#) method.

The following table shows the effect of the different notations:

Example	Type of reference	Description
C5	Relative	Reference is to the relative location of cell C5, depending on the location of the cell in which the reference is first used
\$C\$5	Absolute	Reference is absolute. Will always refer to cell C5 no matter where it is used.
\$C5	Mixed	Reference is always to column C, but the row reference is relative to the location of the cell in which the reference is first used.
C\$5	Mixed	Reference is always to row 5, but the column reference is relative to the location of the cell in which the reference is first used
Cell name	Absolute	Reference is absolute. Will always refer to the <a href="#">named cell or range</a> no matter where the reference is used.

## Built-in functions

Spreadsheet functions are preset formulas used to calculate cell values. When you type the first letter of the function to enter, a pop-up menu listing the available functions and references appears, allowing you to select the desired elements:



See [SpreadJS's extented list of functions](#) for details and examples.

## 4D functions

4D View Pro allows you to define and call **4D custom functions**, which execute [4D formulas](#). Using 4D custom functions extends the possibilities of your 4D View Pro documents and allows powerful interactions with the 4D database.

4D custom functions provide access, from within your 4D View Pro formulas, to:

- 4D process variables,
- fields,
- project methods,
- 4D language commands,
- or any valid 4D expression.

4D custom functions can receive [parameters](#) from the 4D View Pro area, and return values.

You declare all your functions using the [VP\\_SET\\_CUSTOM\\_FUNCTIONS](#) method.  
Examples:

```
o:=New object

//Name of the function in 4D View Pro: "DRIVERS_LICENCE"
$o.DRIVERS_LICENCE:=New object

//process variable
$o.DRIVERS_LICENCE.formula:=Formula(DriverLicence)

//table field
$o.DRIVERS_LICENCE.formula:=Formula([Users]DriverLicence)

//project method
$o.DRIVERS_LICENCE.formula:=Formula(DriverLicenceState)

//4D command
$o.DRIVERS_LICENCE:=Formula(Choose(DriverLicence; "Obtained";
"Failed"))

//4D expression and parameter
$o.DRIVERS_LICENCE.formula:=Formula(ds.Users.get($1).DriverLicence)
$o.DRIVERS_LICENCE.parameters:=New collection
$o.DRIVERS_LICENCE.parameters.push(New object("name"; "ID"; "type"; Is
longint))
```

**See also** [4D View Pro: Use 4D formulas in your spreadsheet \(blog post\)](#)

## Hello World example

We want to print "Hello World" in a 4D View Pro area cell using a 4D project method:

1. Create a "myMethod" project method with the following code:

```
#DECLARE->$hw Text  
$hw:="Hello World"
```

2. Execute the following code before opening any form that contains a 4D View Pro area:

```
Case of  
  :(Form event code=On Load)  
    var $o : Object  
    $o:=New object  
    // Define "vpHello" function from the "myMethod" method  
    $o.vpHello:=New object  
    $o.vpHello.formula:=Formula (myMethod)  
    VP SET CUSTOM FUNCTIONS ("ViewProArea";$o)  
End case
```

3. Edit the content of a cell in a 4D View Pro area and type:



"myMethod" is then called by 4D and the cell displays:



## Parameters

Parameters can be passed to 4D functions that call project methods using the following syntax:

```
=METHODNAME (param1,param2,...,paramN)
```

These parameters are received in *methodName* in \$1, \$2...\$N.

Note that the ( ) are mandatory, even if no parameters are passed:

```
=METHODWITHOUTNAME ()
```

You can declare the name, type, and number of parameters through the *parameters* collection of the function you declared using the [VP SET CUSTOM FUNCTIONS](#) method. Optionally, you can control the number of parameters passed by the user through *minParams* and *maxParams* properties.

For more information on supported incoming parameter types, please refer to the [VP SET CUSTOM FUNCTIONS](#) method description.

If you do not declare parameters, values can be sequentially passed to methods (they will be received in \$1, \$2...) and their type will be automatically converted. Dates in *jstype* will be passed as [object](#) in 4D code with two properties:

Property	Type	Description	---	---	---	value	Date	Date value	time
Real	Time in seconds								

4D project methods can also return values in the 4D View Pro cell formula via \$0. The following data types are supported for returned parameters:

- [text](#) (converted to string in 4D View Pro)
- [real/longint](#) (converted to number in 4D View Pro)

- [date](#) (converted to JS Date type in 4D View Pro - hour, minute, sec = 0)
- [time](#) (converted to JS Date type in 4D View Pro - date in base date, i.e. 12/30/1899)
- [boolean](#) (converted to bool in 4D View Pro)
- [picture](#) (jpg,png,gif,bmp,svg other types converted into png) creates a URI (data:image/png;base64,xxxx) and then used as the background in 4D View Pro in the cell where the formula is executed
- [object](#) with the following two properties (allowing passing a date and time):

**Property Type Description**

value	Date	Date value
time	Real	Time in seconds

If the 4D method returns nothing, an empty string is automatically returned.

An error is returned in the 4D View Pro cell if:

- the 4D method returns another type other than those listed above,
- an error occurred during 4D method execution (when user clicks on "abort" button).

## Example

```
var $o : Object

$o.BIRTH_INFORMATION:=New object
$o.BIRTH_INFORMATION.formula:=Formula(BirthInformation)
$o.BIRTH_INFORMATION.parameters:=New collection
$o.BIRTH_INFORMATION.parameters.push(New object("name";"First
name";"type";Is text))
$o.BIRTH_INFORMATION.parameters.push(New
object("name";"Birthday";"type";Is date))
$o.BIRTH_INFORMATION.parameters.push(New object("name";"Time of
birth";"type";Is time))
$o.BIRTH_INFORMATION.summary:="Returns a formatted string from given
information"

VP SET CUSTOM FUNCTIONS ("ViewProArea"; $o)
```

## Compatibility

Alternate solutions are available to declare fields or methods as functions in your 4D View Pro areas. These solutions are maintained for compatibility reasons and can be used in specific cases. However, using the [VP\\_SET\\_CUSTOM\\_FUNCTIONS](#) method is recommended.

## Referencing fields using the virtual structure

4D View Pro allows you to reference 4D fields using the virtual structure of the database, i.e. declared through the [SET\\_TABLE\\_TITLES](#) and/or [SET\\_FIELD\\_TITLES](#) commands with the \* parameter. This alternate solution could be useful if your application already relies on a virtual structure (otherwise, [using VP\\_SET\\_CUSTOM\\_FUNCTIONS](#) is recommended).

**WARNING:** You cannot use the virtual structure and [VP\\_SET\\_CUSTOM\\_FUNCTIONS](#) simultaneously. As soon as [VP\\_SET\\_CUSTOM\\_FUNCTIONS](#) is called, the functions based upon [SET\\_TABLE\\_TITLES](#) and [SET\\_FIELD\\_TITLES](#) commands are ignored in the 4D View Pro area.

## Requirements

- The field must belong to the virtual structure of the database, i.e. it must be declared through the [SET TABLE TITLES](#) and/or [SET FIELD TITLES](#) commands with the \* parameter (see example),
- Table and field names must be ECMA compliant (see [ECMA Script standard](#)),
- The field type must be supported by 4D View Pro (see above).

An error is returned in the 4D View Pro cell if the formula calls a field which is not compliant.

### Calling a virtual field in a formula

To insert a reference to a virtual field in a formula, enter the field with the following syntax:

```
TABLENAME_FIELDNAME()
```

For example, if you declared the "Name" field of the "People" table in the virtual structure, you can call the following functions:

```
=PEOPLE_NAME()  
=LEN(PEOPLE_NAME())
```

If a field has the same name as a [4D method], it takes priority over the method.

### Example

We want to print the name of a person in a 4D View Pro area cell using a 4D virtual field:

1. Create an "Employee" table with a "L\_Name" field:
2. Execute the following code to initialize a virtual structure:

```
ARRAY TEXT($tableTitles;1)  
ARRAY LONGINT($tableNum;1)  
$tableTitles{1}:="Emp"  
$tableNum{1}:=2  
SET TABLE TITLES($tableTitles;$tableNum;*)  
  
ARRAY TEXT($fieldTitles;1)  
ARRAY LONGINT($fieldNum;1)  
$fieldTitles{1}:="Name"  
$fieldNum{1}:=2 //last name  
SET FIELD TITLES([Employee];$fieldTitles;$fieldNum;*)
```

3. Edit the content of a cell in the 4D View Pro area and enter "=e":

	A	B	C	D	E
1	=e				
2	EDATE				
3	EFFECT				
4	EMP_NAME				
5	ENCODEURL				

4. Select EMP\_NAME (use the Tab key) and enter the closing ).

	A	B
1	=EMP_NAME()	

5. Validate the field to display the name of the current employee:

	A	B
1	Smith	

The [Employee] table must have a current record.

## Declaring allowed methods

You can call directly 4D project methods from within your 4D View Pro formulas. For security reasons, you must declare explicitly methods that can be called by the user with the [VP SET ALLOWED METHODS](#) method.

### Requirements

To be called in a 4D View Pro formula, a project method must be:

- **Allowed:** it was explicitly declared using the [VP SET ALLOWED METHODS](#) method.
- **Runnable:** it belongs to the host project or a loaded component with the "Shared by components and host project" option enabled (see [Sharing of project methods](#)).
- **Not in conflict** with an existing 4D View Pro spreadsheet function: if you call a project method with the same name as a 4D View Pro built-in function, the function is called.

If neither the [VP SET CUSTOM FUNCTIONS](#) nor the [VP SET ALLOWED METHODS](#) method has been executed during the session, 4D View Pro custom functions rely on allowed methods defined by 4D's generic `SET ALLOWED METHODS` command. In this case, the project method names must comply with JavaScript Identifier Grammar (see [ECMA Script standard](#)). The global filtering option in the Settings dialog box (see *Data Access*) is ignored in all cases.

# Method List

**Warning:** The commands on this page are not thread-safe.

[A](#) - [C](#) - [D](#) - [E](#) - [G](#) - [I](#) - [M](#) - [N](#) - [Q](#) - [P](#) - [R](#) - [S](#)

## A

### VP ADD FORMULA NAME

**VP ADD FORMULA NAME** ( *vpAreaName* : Text ; *vpFormula* : Text ; *name* : Text { ; *options* : Object } )

Parameter	Type	Description
<i>vpAreaName</i>	Text	-> 4D View Pro area form object name
<i>vpFormula</i>	Text	-> 4D View Pro formula
<i>name</i>	Text	-> Name for the formula
<i>options</i>	Object	-> Options for the named formula

#### Description

The `VP ADD FORMULA NAME` command creates or modifies a named formula in the open document.

Named formulas created by this command are saved with the document.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

Pass the 4D View Pro formula that you want to name in *vpFormula*. For detailed information about formula syntax, see [Formulas and Functions](#) page.

Pass the new name for the formula in *name*. If the name is already used within the same scope, the new named formula replaces the existing one. Note that you can use the same name for different scopes (see below).

You can pass an object with additional properties for the named formula in *options*. The following properties are supported:

Property	Type	Description
		Scope for the formula. You can pass the sheet index (counting begins at 0) or use the following constants: <ul style="list-style-type: none"><li>• <code>vk current sheet</code></li></ul>
<i>scope</i>	Number	• <code>vk workbook</code> The scope determines whether a formula name is local to a given worksheet ( <i>scope</i> =sheet index or <code>vk current sheet</code> ), or global across the entire workbook ( <i>scope</i> = <code>vk workbook</code> ).
<i>commentText</i>	Comment	associated to named formula

#### Example

```
VP ADD FORMULA NAME ("ViewProArea"; "SUM($A$1:$A$10)"; "Total2")
```

#### See also

[Cell references](#)

[VP Get formula by name](#)

[VP Get names](#)

## VP ADD RANGE NAME

**VP ADD RANGE NAME** ( *rangeObj* : Object ; *name* : Text { ; *options* : Object } )

Parameter	Type	Description
<i>rangeObj</i>	Object->Range object	
<i>name</i>	Text	-> Name for the formula
<i>options</i>	Object->	Options for the named formula

### Description

The `VP ADD RANGE NAME` command creates or modifies a named range in the open document.

Named ranges created by this command are saved with the document.

In *rangeObj*, pass the range that you want to name and in *name*, pass the new name for the range. If the name is already used within the same scope, the new named range replaces the existing one. Note that you can use the same name for different scopes (see below).

You can pass an object with additional properties for the named range in *options*. The following properties are supported:

Property	Type	Description
		Scope for the range. You can pass the sheet index (counting begins at 0) or use the following constants: <ul style="list-style-type: none"><li>• <code>vk current sheet</code></li></ul>
<i>scope</i>	Number	• <code>vk workbook</code> The scope determines whether a range name is local to a given worksheet ( <i>scope</i> =sheet index or <code>vk current sheet</code> ), or global across the entire workbook ( <i>scope</i> = <code>vk workbook</code> ).
<i>commentText</i>	Comment	Associated to named range <ul style="list-style-type: none"><li>• A named range is actually a named formula containing coordinates. <code>VP ADD RANGE NAME</code> facilitates the creation of named ranges, but you can also use the <a href="#">VP ADD FORMULA NAME</a> method to create named ranges.</li><li>• Formulas defining named ranges can be retrieved with the <a href="#">VP Get formula by name</a> method.</li></ul>

### Example

You want to create a named range for a cell range:

```
$range:=VP Cell("ViewProArea";2;10)
VP ADD RANGE NAME($range;"Total1")
```

### See also

[VP Get names](#)

[VP Name](#)

## VP ADD SELECTION

**VP ADD SELECTION** ( *rangeObj* : Object )

### Parameter Type    Description

*rangeObj*   Text -> Range object

### Description

The `VP ADD SELECTION` command adds the specified cells to the currently selected cells.

In *rangeObj*, pass a range object of cells to add to the current selection.

The active cell is not modified.

### Example

You have cells currently selected:

The following code will add cells to your selection:

```
$currentSelection:=VP Cells("myVPArea";3;4;2;3)
VP ADD SELECTION($currentSelection)
```

Result:

### See also

[VP Get active cell](#)

[VP Get selection](#)

[VP RESET SELECTION](#)

[VP SET ACTIVE CELL](#)

[VP SET SELECTION](#)

[VP SHOW CELL](#)

## VP ADD SHEET

**VP ADD SHEET** ( *vpAreaName* : Text )

**VP ADD SHEET** ( *vpAreaName* : Text ; *index* : Integer )

**VP ADD SHEET** ( *vpAreaName* : Text ; *sheet* : Integer ; *name* : Text )

### Parameter Type                      Description

*vpAreaName* Text   -> 4D View Pro area form object  
name

*sheet*        Integer->Index of the new sheet

*name*        Text   -> Sheet name

### Description

The `VP ADD SHEET` command inserts a sheet in the document loaded in *vpAreaName*.

In *vpAreaName*, pass the name of the 4D View Pro area.

In *sheet*, you can pass an index for the new sheet. If the passed *index* is inferior to or

equal to 0, the command inserts the new sheet at the beginning. If *index* exceeds the number of sheets, the command inserts the new sheet after the existing ones.

Indexing starts at 0.

In *name*, you can pass a name for the new sheet. The new name cannot contain the following characters: \*, :, [, ], ?, \, /

## Example

The document currently has 3 sheets:



To insert a sheet at the third position (index 2) and name it "March":

```
VP ADD SHEET("ViewProArea"; 2; "March")
```



## See also

[VP REMOVE SHEET](#)

[VP ADD SPAN](#)

**VP ADD SPAN** ( *rangeObj* : Object )

**Parameter Type Description**

*rangeObj* Object->Range object

## Description

The `VP ADD SPAN` command combines the cells in *rangeObj* as a single span of cells.

In *rangeObj*, pass a range object of cells. The cells in the range are joined to create a larger cell extending across multiple columns and/or rows. You can pass multiple cell ranges to create several spans at the same time. Note that if cell ranges overlap, only the first cell range is used.

- Only the data in the upper-left cell is displayed. Data in the other combined cells is hidden until the span is removed.
- Hidden data in spanned cells is accessible via formulas (beginning with the upper-left cell).

## Example

To span the First quarter and Second quarter cells across the two cells beside them, and the South area cell across the two rows below it:

```

// First quarter range
$q1:=VP Cells("ViewProArea";2;3;3;1)

// Second quarter range
$q2:=VP Cells("ViewProArea";5;3;3;1)

// South area range
$south:=VP Cells("ViewProArea";0;5;1;3)

VP ADD SPAN(VP Combine ranges($q1;$q2;$south) )

```

## See also

[4D View Pro Range Object Properties](#)

[VP Get spans](#)

[VP REMOVE SPAN](#)

## VP ADD STYLESHEET

**VP ADD STYLESHEET** ( *vpAreaName* : Text ; *styleName* : Text ; *styleObj* : Object { ; *sheet* : Integer } )

Parameter	Type	Description
<i>vpAreaName</i>	Text	->4D View Pro area form object name
<i>styleName</i>	Text	->Name of style
<i>styleObj</i>	Object	->Object defining attribute settings
<i>sheet</i>	Integer	->Sheet index (current sheet if omitted)

## Description

The **VP ADD STYLESHEET** command creates or modifies the *styleName* style sheet based upon the combination of the properties specified in *styleObj* in the open document. If a style sheet with the same name and index already exists in the document, this command will overwrite it with the new values.

Style sheets created by this command are saved with the document.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

The *styleName* parameter lets you assign a name to the style sheet. If the name is already used within the same scope, the new style sheet replaces the existing one. Note that you can use the same name for different scopes (see below).

Within the *styleObj*, designate the settings for the style sheet (e.g., font, text decoration, alignment, borders, etc.). For the full list of style properties, see [Style object properties](#).

You can designate where to define the style sheet in the optional *sheet* parameter using the sheet index (indexing starts at 0) or with the following constants:

- **vk current sheet**
- **vk workbook**

If a *styleName* style sheet is defined at the workbook level and at a sheet level, the sheet level has priority over the workbook level when the style sheet is set.

To apply the style sheet, use the [VP SET DEFAULT STYLE](#) or [VP SET CELL STYLE](#) commands.

## Example

The following code:

```
$styles:=New object
$styles.backColor:="green"

//Line Border Object
$borders:=New object("color";"green";"style";vk line style medium dash
dot)

$styles.borderBottom:=$borders
$styles.borderLeft:=$borders
$styles.borderRight:=$borders
$styles.borderTop:=$borders

VP ADD STYLESHEET("ViewProArea";"GreenDashDotStyle";$styles)

//To apply the style
VP SET CELL STYLE(VP Cells("ViewProArea";1;1;2;2);New
object("name";"GreenDashDotStyle"))
```

will create and apply the following style object named *GreenDashDotStyle*:

```
{
  backColor:green,
  borderBottom:{color:green,style:10},
  borderLeft:{color:green,style:10},
  borderRight:{color:green,style:10},
  borderTop:{color:green,style:10}
}
```

## See also

[4D View Pro Style Objects and Style Sheets](#)  
[VP Get stylesheet](#)  
[VP Get stylesheets](#)  
[VP REMOVE STYLESHEET](#)  
[VP SET CELL STYLE](#)  
[VP SET DEFAULT STYLE](#)

## VP All

**VP All** ( *vpAreaName* : Text { ; *sheet* : Integer } ) : Object

Parameter	Type	Description
<i>vpAreaName</i>	Text	->4D View Pro area form object name
<i>sheet</i>	Integer	-> Sheet index (current sheet if omitted)
Result	Object	<-Range object of all cells

## Description

The **VP ALL** command returns a new range object referencing all cells.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

In the optional *sheet* parameter, you can designate a specific spreadsheet where the range will be defined (counting begins at 0). If omitted or if you pass `vk current sheet`, the current spreadsheet is used.

## Example

You want to define a range object for all of the cells of the current spreadsheet:

```
$all:=VP All("ViewProArea") // all cells of the current sheet
```

## See also

[VP Cell](#)  
[VP Cells](#)  
[VP Column](#)  
[VP Combine ranges](#)  
[VP Name](#)  
[VP Row](#)

## C

### VP Cell

**VP Cell** ( *vpAreaName* ; *column* : Integer ; *row* : Integer ; Text { ; *sheet* : Integer } )  
: Object

Parameter	Type	Description
<i>vpAreaName</i>	Text	-> 4D View Pro area form object name
<i>column</i>	Longint	-> Column index
<i>row</i>	Longint	-> Row index
<i>sheet</i>	Integer	-> Sheet index (current sheet if omitted)
Result	Object	<- Range object of a single cell

### Description

The `VP Cell` command returns a new range object referencing a specific cell.

This command is intended for ranges of a single cell. To create a range object for multiple cells, use the [VP Cells](#) command.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

The *column* parameter defines the column of the cell range's position. Pass the column index in this parameter.

The *row* parameter defines the row of the cell range's position. Pass the row index in this parameter.

In the optional *sheet* parameter, you can indicate the index of the sheet where the range will be defined. If omitted or if you pass `vk current sheet`, the current spreadsheet is used by default.

indexing starts at 0.

## Example

You want to define a range object for the cell shown below (on the current spreadsheet):

	A	B	C	D	E	F
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						

The code would be:

```
$cell:=VP Cell("ViewProArea";2;4) // C5
```

## See also

[VP All](#)  
[VP Cells](#)  
[VP Column](#)  
[VP Combine ranges](#)  
[VP Name](#)  
[VP Row](#)

## VP Cells

**VP Cells** ( *vpAreaName* : Text ; *column*: Integer ; *row*: Integer ; *columnCount* : Integer ; *rowCount* : Integer { ; *sheet* : Integer } ) : Object

- History

Parameter	Type	Description
<i>vpAreaName</i>	Text	->4D View Pro area form object name
<i>column</i>	Integer->Column index	
<i>row</i>	Integer->Row index	
<i>columnCount</i>	Integer->Number of columns	
<i>rowCount</i>	Integer->Number of rows	
<i>sheet</i>	Integer->	Sheet index (current sheet if omitted)
<i>Result</i>	Object	<-Range object of cells

## Description

The `VP Cells` command returns a new range object referencing specific cells.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

The *column* parameter defines the first column of the cell range. Pass the column index (counting begins at 0) in this parameter. If the range is within multiple columns, you should also use the *columnCount* parameter.

In the *row* parameter, you can define the row(s) of the cell range's position. Pass the row index (counting begins at 0) in this parameter. If the range is within multiple

rows, you should also use the *rowCount* parameter.

The *columnCount* parameter allows you to define the total number of columns the range is within. *columnCount* must be greater than 0.

The *rowCount* parameter allows you to define the total number of rows the range is within. *rowCount* must be greater than 0.

In the optional *sheet* parameter, you can designate a specific spreadsheet where the range will be defined (counting begins at 0). If omitted or if you pass `vk current sheet`, the current spreadsheet is used by default.

## Example

You want to define a range object for the following cells (on the current sheet):

The code would be:

```
$cells:=VP Cells("ViewProArea";2;4;2;3) // C5 to D7
```

## See also

[VP All](#)

[VP Cells](#)

[VP Column](#)

[VP Combine ranges](#)

[VP Name](#)

[VP Row](#)

## VP Column

**VP Column** (*vpAreaName* : Text ; *column*: Integer ; *columnCount* : Integer { ; *sheet* : Integer } ) : Object

Parameter	Type	Description
<i>vpAreaName</i>	Text	->4D View Pro area form object name
<i>column</i>	Integer	->Column index
<i>columnCount</i>	Integer	->Number of columns
<i>sheet</i>	Integer	-> Sheet index (current sheet if omitted)
<i>Result</i>	Object	<-Range object of cells

## Description

The `VP Column` command returns a new range object referencing a specific column or columns.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

The *column* parameter defines the first column of the column range. Pass the column index (counting begins at 0) in this parameter. If the range contains multiple columns, you should also use the optional *columnCount* parameter.

The optional *columnCount* parameter allows you to define the total number of columns of the range. *columnCount* must be greater than 0. If omitted, the value will be set to

1 by default and a column type range is created.

In the optional `sheet` parameter, you can designate a specific spreadsheet where the range will be defined (counting begins at 0). If omitted or if you pass `vk current sheet`, the current spreadsheet is used by default.

## Example

You want to define a range object for the column shown below (on the current spreadsheet):

The code would be:

```
$column:=VP Column("ViewProArea";3) // column D
```

## See also

[VP All](#)  
[VP Cells](#)  
[VP Column](#)  
[VP Combine ranges](#)  
[VP Name](#)  
[VP Row](#)  
[VP SET COLUMN ATTRIBUTES](#)

## VP COLUMN AUTOFIT

**VP COLUMN AUTOFIT** ( *rangeObj* : Object )

**Parameter Type Description**  
*rangeObj* Object->Range object

### Description

The `VP COLUMN AUTOFIT` command automatically sizes the column(s) in *rangeObj* according to their contents.

In *rangeObj*, pass a range object containing a range of the columns whose size will be automatically handled.

## Example

The following columns are all the same size and don't display some of the text:

	A	B	C	D	E
1		Hello Wor	The quick	TGIF	
2					
3					
4					
5					

Selecting the columns and running this code:

```
VP COLUMN AUTOFIT(VP Get selection("ViewProarea"))
```

... resizes the columns to fit the size of the contents:

	A	B	C	D	E
1					
2					
3					
4					
c		Hello World	The quick brown fox jumped over the lazy dog. TGI		

## See also

[VP ROW AUTOFIT](#)

## VP Combine ranges

**VP Combine ranges** ( *rangeObj* : Object ; *otherRangeObj* : Object {;...*otherRangeObjN* : Object } ) : Object

Parameter	Type	Description
<i>rangeObj</i>	Object -> Range object	
<i>otherRangeObj</i>	Object -> Range object	
<i>Result</i>	Object <- Object containing a combined range	

## Description

The `VP Combine Ranges` command returns a new range object that incorporates two or more existing range objects. All of the ranges must be from the same 4D View Pro area.

In *rangeObj*, pass the first range object.

In *otherRangeObj*, pass another range object(s) to combine with *rangeObj*.

The command incorporates *rangeObj* and *otherRangeObj* objects by reference.

## Example

You want to combine cell, column, and row range objects in a new, distinct range object:

```
$cell:=VP Cell("ViewProArea";2;4) // C5
$column:=VP Column("ViewProArea";3) // column D
$row:=VP Row("ViewProArea";9) // row 10

$combine:=VP Combine ranges($cell;$column;$row)
```

## See also

[VP All](#)  
[VP Cells](#)  
[VP Column](#)  
[VP Combine ranges](#)  
[VP Name](#)  
[VP Row](#)  
[VP SET COLUMN ATTRIBUTES](#)

## VP Convert from 4D View

**VP Convert from 4D View** ( *4DViewDocument* : Blob ) : Object

Parameter	Type	Description
4DViewDocument	Blob	-> 4D View document
Result	Object	<- 4D View Pro object

## Description

The `VP Convert from 4D View` command allows you to convert a legacy 4D View document into a 4D View Pro object.

This command does not require that the legacy 4D View plug-in be installed in your environment.

In the *4DViewDocument* parameter, pass a BLOB variable or field containing the 4D View document to convert. The command returns a 4D View Pro object into which all the information originally stored within the 4D View document is converted to 4D View Pro attributes.

## Example

You want to get a 4D View Pro object from a 4D View area stored in a BLOB:

```
C_OBJECT($vpObj)
$vpObj:=VP Convert from 4D View($pvcblob)
```

## VP Convert to picture

**VP Convert to picture** (*vpObject* : Object {; *rangeObj* : Object} ) : Picture

Parameter	Type	Description
<i>vpObject</i>	Object->	4D View Pro object containing the area to convert
<i>rangeObj</i>	Object->	Range object
Result	Object<-	SVG picture of the area

## Description

The `VP Convert to picture` command converts the *vpObject* 4D View Pro object (or the *rangeObj* range within *vpObject*) to a SVG picture.

This command is useful, for example:

- to embed a 4D View Pro document in an other document such as a 4D Write Pro document
- to print a 4D View Pro document without having to load it into a 4D View Pro area.

In *vpObject*, pass the 4D View Pro object that you want to convert. This object must have been previously parsed using [VP Export to object](#) or saved using [VP EXPORT DOCUMENT](#).

SVG conversion process requires that expressions and formats (cf. [Cell Format](#)) included in the 4D View Pro area be evaluated at least once, so that they can be correctly exported. If you convert a document that was not evaluated beforehand, expressions or formats may be rendered in an unexpected way.

In *rangeObj*, pass a range of cells to convert. By default, if this parameter is omitted, the whole document contents are converted.

Document contents are converted with respect to their viewing attributes, including formats (see note above), visibility of headers, columns and rows. The conversion of the following elements is supported:

- Text : style / font / size / alignment / orientation / rotation / format
- Cell background : color / image
- Cell borders : thickness / color / style
- Cell merge
- Pictures
- Row height
- Column width
- Hidden columns / rows.

Gridline visibility depends on document attribute defined with [VP SET PRINT INFO](#).

## Function result

The command returns a picture in SVG format.

## Example

You want to convert a 4D View Pro area in SVG, preview the result, and send it to a picture variable:

```
C_OBJECT($vpAreaObj)
C_PICTURE($vPict)
$vpAreaObj:=VP Export to object("ViewProArea")
$vPict:=VP Convert to picture($vpAreaObj) //export the whole area
```

## See also

[VP EXPORT DOCUMENT](#)  
[VP Export to object](#)  
[VP SET PRINT INFO](#)

## VP Copy to object

- History

**VP Copy to object** (*rangeObj* : Object {; *options* : Object} ) : Object

Parameter	Type	Description
rangeObj	Object->Range object	
options	Object->Additional options	
Result	Object<-	Object returned. Contains the copied data

## Description

The `VP Copy to object` command copies the contents, style and formulas from *rangeObj* to an object.

In *rangeObj*, pass the cell range with the values, formatting, and formulas to copy. If *rangeObj* is a combined range, only the first one is used.

You can pass an optional *options* parameter with the following properties:

Property	Type	Description														
copy	Boolean	<p><i>True</i> (default) to keep the copied values, formatting and formulas after the command executes. <i>False</i> to remove them.</p> <p>Specifies what is copied or moved. Possible values:</p>														
		<table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>vk clipboard options all (default)</td><td>Copies all data objects, including values, formatting, and formulas.</td></tr> <tr> <td>vk clipboard options formatting</td><td>Copies only the formatting.</td></tr> <tr> <td>vk clipboard options formulas</td><td>Copies only the formulas.</td></tr> <tr> <td>vk clipboard options formulas and formatting</td><td>Copies the formulas and formatting.</td></tr> <tr> <td>vk clipboard options values</td><td>Copies only the values.</td></tr> <tr> <td>vk clipboard options value and formatting</td><td>Copies the values and formatting.</td></tr> </tbody> </table>	Value	Description	vk clipboard options all (default)	Copies all data objects, including values, formatting, and formulas.	vk clipboard options formatting	Copies only the formatting.	vk clipboard options formulas	Copies only the formulas.	vk clipboard options formulas and formatting	Copies the formulas and formatting.	vk clipboard options values	Copies only the values.	vk clipboard options value and formatting	Copies the values and formatting.
Value	Description															
vk clipboard options all (default)	Copies all data objects, including values, formatting, and formulas.															
vk clipboard options formatting	Copies only the formatting.															
vk clipboard options formulas	Copies only the formulas.															
vk clipboard options formulas and formatting	Copies the formulas and formatting.															
vk clipboard options values	Copies only the values.															
vk clipboard options value and formatting	Copies the values and formatting.															
copyOptions	Longint															

The paste options defined in the [workbook options](#) are taken into account.

The command returns an object that contains the copied data.

## Example

This code sample first stores the contents, values, formatting and formulas from a range to an object, and then pastes them in another range:

```
var $originRange; $targetRange; $dataObject; $options : Object
$originRange:=VP Cells("ViewProArea"; 0; 0; 2; 5)

$options:=New object
$options.copy:=True
$options.copyOptions:=vk clipboard options all

$dataObject:=VP Copy to object($originRange; $options)

$targetRange:=VP Cell("ViewProArea"; 4; 0)
VP PASTE FROM OBJECT($targetRange; $dataObject; vk clipboard options all)
```

## See also

[VP PASTE FROM OBJECT](#)  
[VP MOVE CELLS](#)  
[VP Get workbook options](#)  
[VP SET WORKBOOK OPTIONS](#)

## VP CREATE TABLE

- History

**VP CREATE TABLE** ( *rangeObj* : Object ; *tableName* : Text {; *source* : Text} {; *options* : cs.ViewPro.TableOptions} )

Parameter	Type	Description
rangeObj	Object	- > Range object
tableName	Text	- > Name for the table
source	Text	- Data context property name to display in the > table
options	<a href="#">cs.ViewPro.TableOptions</a>	- > Additional options

## Description

The `VP CREATE TABLE` command creates a table in the specified range. You can create a table in a range of cells to make managing and analyzing a group of related data easier. A table typically contains related data in rows and columns, and takes advantage of a [data context](#).

In *rangeObj*, pass the cell range where the table will be created.

In *tableName*, pass a name for the table. The name must:

- be unique in the sheet
- include at least 5 characters
- not include spaces or start with a number

In *source*, you can pass a property name of a [data context](#) to display its data in the table. This binds the table to the data context. When the data context is updated, the data displayed in the table is updated accordingly. The *source* property must contain a collection of objects and each element represents a row.

- If you don't specify a *source*, the command creates an empty table with the size defined in *rangeObj*.
- If the specified *source* cannot be fully displayed in the document, no table is created.

In the *options* parameter, pass an object of the [cs.ViewPro.TableOptions class](#) that contains the table properties to set.

Within the *options* object, the *tableColumns* collection determines the structure of the table's columns. The length of the *tableColumns* collection must be equal to the range column count:

- When the column count in *rangeObj* exceeds the number of columns in *tableColumns*, the table is filled with additional empty columns.
- When the column count in *rangeObj* is inferior to the number of *tableColumns*, the table displays a number of columns that match the range's column count.

If you pass a *source* but no *tableColumn* option, the command generates columns automatically. In this case, *rangeObj* must be a cell range. Otherwise, the first cell of the range is used. When generating columns automatically, the following rules apply:

- If the data passed to the command is a collection of objects, the property names are used as column titles. For example:

```
([{ LastName: \"Freehafer\", FirstName: \"Nancy\"}, { LastName:  
\"John\", FirstName: \"Doe\"})
```

Here the titles of the columns would be `LastName` and `FirstName`.

- If the data passed to the command is a collection of scalar values, it must contain a collection of subcollections:
  - The first-level collection contains subcollections of values. Each subcollection defines a row. Pass an empty collection to skip a row. The number of values in the first subcollection determines how many columns are created.
  - The subcollections' indices are used as column titles.
  - Each subcollection defines cell values for the row. Values can be `Integer`, `Real`, `Boolean`, `Text`, `Date`, `Null`, `Time` or `Picture`. A `Time` value must be an object containing a `time` attribute, as described in [VP SET VALUE](#).

This only works when generating columns automatically. You cannot use a collection of scalar data with the `tableColumns` option.

## Example

To create a table using a data context:

```
// Set a data context
var $data : Object

$data:=New object()
$data.people:=New collection()
$data.people.push(New object("firstName"; "John"; "lastName"; "Smith";
"email"; "johnsmith@gmail.com"))
$data.people.push(New object("firstName"; "Mary"; "lastName";
"Poppins"; "email"; "marypoppins@gmail.com"))

VP SET DATA CONTEXT("ViewProArea"; $data)

// Define the columns for the table
var $options : cs.ViewPro.TableOptions

$options:=cs.ViewPro.TableOptions.new()
$options.tableColumns:=New collection()
$options.tableColumns.push(cs.ViewPro.TableColumns.new("name"; "First
name"; "dataField"; "firstName"))
$options.tableColumns.push(cs.ViewPro.TableColumns.new("name"; "Last
name"; "dataField"; "lastName"))
$options.tableColumns.push(cs.ViewPro.TableColumns.new("name"; "Email";
"dataField"; "email"))

// Create a table from the "people" collection
VP CREATE TABLE(VP Cells("ViewProArea"; 1; 1;
$options.tableColumns.length; 1); "ContextTable"; "people"; $options)
```

Here's the result:

## See also

[VP Find table](#)  
[VP Get table column attributes](#)  
[VP Get table column index](#)  
[VP INSERT TABLE COLUMNS](#)  
[VP INSERT TABLE ROWS](#)

[VP REMOVE TABLE](#)

[VP RESIZE TABLE](#)

[VP SET DATA CONTEXT](#)

[VP SET TABLE COLUMN ATTRIBUTES](#)

[VP SET TABLE THEME](#)

## D

### VP DELETE COLUMNS

**VP DELETE COLUMNS** (*rangeObj* : Object)

**Parameter Type Description**

rangeObj Object->Range object

#### Description

The `VP DELETE COLUMNS` command removes the columns in the *rangeObj*.

In *rangeObj*, pass an object containing a range of columns to remove. If the passed range contains:

- both columns and rows, only the columns are removed.
- only rows, the command does nothing.

Columns are deleted from right to left.

#### Example

To delete columns selected by the user (in the image below columns B, C, and D):

	A	B	C	D	E	F	G	H	I
1									
2									
3									
4	The	quick	brown	fox	jumped	over	the	lazy	dog
c									

use the following code:

```
VP DELETE COLUMNS(VP Get selection("ViewProArea"))
```

#### See also

[VP All](#)

[VP Cells](#)

[VP Column](#)

### VP DELETE ROWS

**VP DELETE ROWS** (*rangeObj* : Object)

**Parameter Type Description**

rangeObj Object->Range object

#### Description

The `VP DELETE ROWS` command removes the rows in the *rangeObj*.

In *rangeObj*, pass an object containing a range of rows to remove. If the passed range contains:

- both columns and rows, only the rows are removed.
- only columns, the command does nothing.

Rows are deleted from bottom to top.

## Example

To delete rows selected by the user (in the image below rows 1, 2, and 3):

	A	B	C	D	E	F	G	H	I
1									
2									
3	The	quick	brown	fox	jumped	over	the	lazy	dog

use the following code:

```
VP DELETE ROWS(VP Get selection("ViewProArea"))
```

## See also

[VP All](#)  
[VP Cells](#)  
[VP Column](#)

## E

### VP EXPORT DOCUMENT

**VP EXPORT DOCUMENT** ( *vpAreaName* : Text ; *filePath* : Text {; *paramObj* : Object} )

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>vpAreaName</i>	Text	-> 4D View Pro area form object name
<i>filePath</i>	Text	-> Pathname of the document
<i>paramObj</i>	Object	-> Export options

#### Description

The `VP EXPORT DOCUMENT` command exports the 4D View Pro object attached to the 4D View Pro area *vpAreaName* to a document on disk according to the *filePath* and *paramObj* parameters.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

In *filePath*, pass the destination path and name of the document to be exported. If you don't specify a path, the document will be saved at the same level as the Project folder.

You can specify the exported file's format by including an extension after the document's name:

- 4D View Pro ("."4vp")
- Microsoft Excel ("xlsx")
- PDF ("pdf")
- CSV ("txt", or "csv")

If the extension is not included, but the format is specified in *paramObj*, the exported file will have the extension that corresponds to the format, except for the CSV format (no extension is added in this case).

The optional *paramObj* parameter allows you to define multiple properties for the exported 4D View Pro object, as well as launch a callback method when the export has completed.

<b>Property</b>	<b>Type</b>	<b>Description</b>									
format	text	<p>(optional) When present, designates the exported file format: ".4vp" (default), ".csv", ".xlsx", or ".pdf". You can use the following constants:</p> <ul style="list-style-type: none"> <li>• <code>vk 4D View Pro format</code></li> <li>• <code>vk csv format</code></li> <li>• <code>vk MS Excel format</code></li> <li>• <code>vk pdf format</code></li> </ul> <p>4D adds the appropriate extension to the file name if needed. If the format specified doesn't correspond with the extension in <i>filePath</i>, it will be added to the end of <i>filePath</i>. If a format is not specified and no extension is provided in <i>filePath</i>, the default file format is used.</p>									
password	text	Microsoft Excel only (optional) - Password used to protect the MS Excel document									
formula	object	Callback method to be launched when the export has completed. Using a callback method is necessary when the export is asynchronous (which is the case for PDF and Excel formats) if you need some code to be executed after the export. The callback method must be used with the <a href="#">Formula</a> command (see below for more information).									
valuesOnly	boolean	<p>Specifies that only the values from formulas (if any) will be exported.</p> <p>True to include formatting information, false otherwise (default is true). Formatting information is useful in some cases, e.g. for export to SVG. On the other hand, setting this property to <b>false</b> allows reducing export time.</p>									
includeFormatInfo	boolean	4DVP and Microsoft Excel only. True (default) to export the current data context values as cell values in the exported document (data contexts themselves are not exported). False otherwise. Cell binding is always exported. For data context and cell binding management, see <a href="#">VP SET DATA CONTEXT</a> and <a href="#">VP SET BINDING PATH</a> .									
sheet		<p>PDF only (optional) - Index of sheet to export (starting number from 0). -2=all visible sheets (<b>default</b>), -1=current sheet only</p> <p>PDF only (optional) - Options for pdf export</p>									
<b>Property Type</b>		<b>Description</b>									
pdfOptions	object	<table border="1"> <tr> <td>creator</td><td>text</td><td>name of the application that created the original document from which it was converted.</td></tr> <tr> <td></td><td>title</td><td>title of the document.</td></tr> <tr> <td></td><td>author</td><td>name of the person who created that document.</td></tr> </table>	creator	text	name of the application that created the original document from which it was converted.		title	title of the document.		author	name of the person who created that document.
creator	text	name of the application that created the original document from which it was converted.									
	title	title of the document.									
	author	name of the person who created that document.									

<b>Property</b>	<b>Type</b>	<b>Description</b>									
	keywordstext keyword subject text	Keywords and subject with the document.									
	CSV only (optional) - Options for csv export										
<b>Property</b>	<b>Type</b>	<b>Description</b>									
csvOptions	object	<table border="1"> <tr> <td>range</td><td>object</td><td>Range object of cells</td></tr> <tr> <td>rowDelimiter</td><td>text</td><td>Row delimiter. Default: "¥r¥n"</td></tr> <tr> <td>columnDelimiter</td><td>text</td><td>Column delimiter. Default: ","</td></tr> </table>	range	object	Range object of cells	rowDelimiter	text	Row delimiter. Default: "¥r¥n"	columnDelimiter	text	Column delimiter. Default: ","
range	object	Range object of cells									
rowDelimiter	text	Row delimiter. Default: "¥r¥n"									
columnDelimiter	text	Column delimiter. Default: ","									
\<customProperty>	any	Any custom property that will be available through the \$3 parameter in the callback method.									

### Notes about Excel format:

- When exporting a 4D View Pro document into a Microsoft Excel-formatted file, some settings may be lost. For example, 4D methods and formulas are not supported by Excel. You can verify other settings with [this list from GrapeCity](#).
- Exporting in this format is run asynchronously, use the *formula* property of the *paramObj* for code to be executed after the export.

### Notes about PDF format:

- When exporting a 4D View Pro document in PDF, the fonts used in the document are automatically embedded in the PDF file. Only OpenType fonts (.OTF or .TTF files) having a Unicode map can be embedded. If no valid font file is found for a font, a default font is used instead.
- Exporting in this format is run asynchronously, use the *formula* property of the *paramObj* for code to be executed after the export.

### Notes about CSV format:

- When exporting a 4D View Pro document to CSV, some settings may be lost, as only the text and values are saved.
- All the values are saved as double-quoted strings. For more information on delimiter-separated values, see [this article on Wikipedia](#).

Once the export operation is finished, `VP EXPORT DOCUMENT` automatically triggers the execution of the method set in the *formula* property of the *paramObj*, if used.

### Passing a callback method (*formula*)

When including the optional *paramObj* parameter, the `VP EXPORT DOCUMENT` command allows you to use the `Formula` command to call a 4D method which will be executed once the export has completed. The callback method will receive the following values in local variables:

<b>Variable</b>	<b>Type</b>	<b>Description</b>
\$1	text	The name of the 4D View Pro object
\$2	text	The filepath of the exported 4D View Pro object
\$3	object	A reference to the command's <i>paramObj</i>
\$4	object	An object returned by the method with a status message
.success	boolean	True if export with success, False otherwise.
.errorCode	integer	Error code. May be returned by 4D or JavaScript.
.errorMessage	text	Error message. May be returned by 4D or JavaScript.

## **Example 1**

You want to export the contents of the "VPArea" area to a 4D View Pro document on disk:

```
var $docPath: Text  
  
$docPath:="C:\\Bases\\ViewProDocs\\MyExport.4VP"  
VP EXPORT DOCUMENT("VPArea";$docPath)  
//MyExport.4VP is saved on your disk
```

## **Example 2**

You want to export the current sheet in PDF:

```
var $params: Object  
$params:=New object  
$params.format:=vk pdf format  
$params.sheet:=-1  
$params.pdfOptions:=New object("title";"Annual Report";"author";Current user)  
VP EXPORT DOCUMENT("VPArea";"report.pdf";$params)
```

## **Example 3**

You want to export a 4D View Pro document in ".xlsx" format and call a method that will launch Microsoft Excel with the document open once the export has completed:

```
$params:=New object  
$params.formula:=Formula(AfterExport)  
$params.format:=vp MS Excel format //".xlsx"  
$params.valuesOnly:=True  
  
VP EXPORT DOCUMENT("ViewProArea";"c:\\tmp\\convertedfile";$params)
```

**AfterExport** method:

```
C_TEXT($1;$2)  
C_OBJECT($3;$4)  
$areaName:=$1  
$filePath:=$2  
$params:=$3  
$status:=$4  
  
If($status.success=False)  
    ALERT($status.errorMessage)  
Else  
    LAUNCH EXTERNAL PROCESS("C:\\Program Files\\Microsoft Office\\Office15\\excel "+$filePath)  
End if
```

## **Example 4**

You want to export the current sheet to a `.txt` file with pipe-separated values:

```

var $params : Object
$params:=New object
$params.range:=VP Cells("ViewProArea";0;0;2;5)
$params.rowDelimiter:="\n"
$params.columnDelimiter:="|"
VP EXPORT DOCUMENT("ViewProArea";"c:\\tmp\\data.txt";New
object("format";vk csv format;"csvOptions";$params))

```

Here's the result:

```

"Clark"|"Kent"
"Bruce"|"Wayne"
"Barry"|"Allen"
"Peter"|"Parker"
"Tony"|"Stark"

```

## See also

[VP Convert to picture](#)  
[VP Export to object](#)  
[VP Column](#)  
[VP Print](#)

## VP Export to object

**VP Export to object** ( *vpAreaName* : Text {; *options* : Object} ) : Object

Parameter	Type	Description
<i>vpAreaName</i>	Text -> name	4D View Pro area form object
<i>options</i>	Object->Export options	
Result	Object<-4D View Pro object	

## Description

The `VP Export to object` command returns the 4D View Pro object attached to the 4D View Pro area *vpAreaName*. You can use this command for example to store the 4D View Pro area in a 4D database object field.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

In the *options* parameter, you can pass the following export options, if required:

Property	Type	Description
<i>includeFormatInfo</i>	Boolean	True (default) to include formatting information, false otherwise. Formatting information is useful in some cases, e.g. for export to SVG. On the other hand, setting this property to False allows reducing export time.
<i>includeBindingSource</i>	Boolean	True (default) to export the current data context values as cell values in the exported object (data contexts themselves are not exported). False otherwise. Cell binding is always exported.

For more information on 4D View Pro objects, please refer to the [4D View Pro object](#) paragraph.

## Example 1

You want to get the "version" property of the current 4D View Pro area:

```
var $vpAreaObj : Object
var $vpVersion : Number
$vpAreaObj:=VP Export to object("vpArea")
// $vpVersion:=OB Get($vpAreaObj;"version")
$vpVersion:=$vpAreaObj.version
```

## Example 2

You want to export the area, excluding formatting information:

```
var $vpObj : Object
$vpObj:=VP Export to object("vpArea";New
object("includeFormatInfo";False))
```

## See also

[VP Convert to picture](#)  
[VP EXPORT DOCUMENT](#)  
[VP IMPORT FROM OBJECT](#)

## F

### VP Find

**VP Find** ( *rangeObj* : Object ; *searchValue* : Text ) : Object  
**VP Find** ( *rangeObj* : Object ; *searchValue* : Text ; *searchCondition* : Object } ) : Object  
**VP Find** ( *rangeObj* : Object ; *searchValue* : Text ; *searchCondition* : Object ; *replaceValue* : Text ) : Object

Parameter	Type	Description
<i>rangeObj</i>	Object->Range object	
<i>searchValue</i>	Text -> Search value	
<i>searchCondition</i>	Object -> Object containing search condition(s)	
<i>replaceValue</i>	Text -> Replacement value	
<i>Result</i>	Object<-Range object	

### Description

The `VP Find` command searches the *rangeObj* for the *searchValue*. Optional parameters can be used to refine the search and/or replace any results found.

In the *rangeObj* parameter, pass an object containing a range to search.

The *searchValue* parameter lets you pass the text to search for within the *rangeObj*.

You can pass the optional *searchCondition* parameter to specify how the search is performed. The following properties are supported:

Property	Type	Description
afterColumnInteger		The number of the column just before the starting column of the search. If the <i>rangeObj</i> is a combined range, the column number given must be from the first range. Default value: -1 (beginning of the <i>rangeObj</i> )
afterRow	Integer	The number of the row just before the starting row of the search. If the <i>rangeObj</i> is a combined range, the row number given must be from the first range. Default value: -1 (beginning of the <i>rangeObj</i> )
all	Boolean	<ul style="list-style-type: none"> <li>True - All cells in <i>rangeObj</i> corresponding to <i>searchValue</i> are returned</li> <li>False - (default value) Only the first cell in <i>rangeObj</i> corresponding to <i>searchValue</i> is returned</li> </ul>
flags	Integer	<pre> vk find flag exact exact match vk find flag ignore ignore case vk find flag none no search flags are considered (default) </pre> <p>Wildcard characters (*,?) can be used in the search string. Wildcard characters can be used in any string comparison to match any number of characters:</p> <ul style="list-style-type: none"> <li>* for zero or multiple characters (for example, searching for "bl*" can find "bl", "black", or "blob")</li> <li>? for a single character (for example, searching for "h?t" can find "hot", or "hit")</li> </ul> <p>These flags can be combined. For example: <code>\$search.flags:=vk find flag use wild cards+vk find flag ignore case</code></p>
order	Integer	<pre> vk find order by columns columns vk find order by rows rows </pre> <p>The search is performed by columns. Each row of a column is searched before the search continues to the next column.</p> <p>The search is performed by rows. Each column of a row is searched before the search continues to the next row (default)</p>
target	Integer	<pre> vk find target formula vk find target tag vk find target text </pre> <p>The search is performed in the cell formula</p> <p>The search is performed in the cell tag</p> <p>The search is performed in the cell text (default)</p> <p>These flags can be combined. For example: <code>\$search.target:=vk find target formula+vk find target tag</code></p>

In the optional *replaceValue* parameter, you can pass text to take the place of any instance of the text in *searchValue* found in the *rangeObj*.

## Returned Object

The function returns a range object describing each search value that was found or

replaced. An empty range object is returned if no results are found.

## Example 1

To find the first cell containing the word "Total":

```
var $range;$result : Object  
$range:=VP All("ViewProArea")  
$result:=VP Find($range;"Total")
```

## Example 2

To find "Total" and replace it with "Grand Total":

```
var $range;$condition;$result : Object  
$range:=VP All("ViewProArea")  
  
$condition:=New object  
$condition.target:=vk find target text  
$condition.all:=True //Search entire document  
$condition.flags:=vk find flag exact match  
  
// Replace the cells containing only 'Total' in the current sheet  
with "Grand Total"  
$result:=VP Find($range;"Total";$condition;"Grand Total")  
  
// Check for empty range object  
If($result.ranges.length=0)  
    ALERT("No result found")  
Else  
    ALERT($result.ranges.length+" results found")  
End if
```

## VP Find table

- History

**VP Find table** ( *rangeObj* : Object ) : Text

### Parameter Type Description

rangeObj Object->Cell range

Result Text <-Table name

### Description

The `VP Find table` command returns the name of the table to which the *rangeObj* cell belongs.

In *rangeObj*, pass a cell range object. If the designated cells do not belong to a table, the command returns an empty string.

If *rangeObj* is not a cell range or contains multiple ranges, the first cell of the first range is used.

## Example

```

If (FORM Event.code=On After Edit && FORM Event.action="valueChanged")
    $tableName:=VP Find table(FORM Event.range)
    If ($tableName!="")
        ALERT("The "+$tableName+" table has been modified.")
    End if
End if

```

## See also

[VP Get table range](#)

## VP FLUSH COMMANDS

**VP FLUSH COMMANDS** ( *vpAreaName* : Text )

Parameter	Type	Description
<i>vpAreaName</i>	Text -> 4D View Pro area form object name	

### Description

The **VP FLUSH COMMANDS** command immediately executes stored commands and clears the command buffer.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

In order to increase performance and reduce the number of requests sent, the 4D View Pro commands called by the developer are stored in a command buffer. When called, **VP FLUSH COMMANDS** executes the commands as a batch when leaving the method and empties the contents of the command buffer.

### Example

You want to trace the execution of the commands and empty the command buffer:

```

VP SET TEXT VALUE(VP Cell("ViewProArea1";10;1);"INVOICE")
VP SET TEXT VALUE(VP Cell("ViewProArea1";10;2);"Invoice date: ")
VP SET TEXT VALUE(VP Cell("ViewProArea1";10;3);"Due date: ")

VP FLUSH COMMANDS ( ("ViewProArea1")
TRACE

```

## VP Font to object

**VP Font to object** ( *font* : Text ) : Object

Parameter	Type	Description
<i>font</i>	Text -> Font shorthand string	
Result	Object <- Font object	

### Description

The **VP Font to object** utility command returns an object from a font shorthand string. This object can then be used to set or get font property settings via object notation.

In the *font* parameter, pass a font shorthand string to specify the different properties

of a font (e.g., "12 pt Arial"). You can learn more about font shorthand strings [in this page](#) for example.

The returned object contains defined font attributes as properties. For more information about the available properties, see the [VP Object to font](#) command.

## Example 1

This code:

```
$font:=VP Font to object ("16pt arial")
```

will return the following \$font object:

```
{  
  
family:arial  
size:16pt  
}
```

## Example 2

See example for [VP Object to font](#).

## See also

[4D View Pro Style Objects and Style Sheets](#)

[VP Object to font](#)

[VP SET CELL STYLE](#)

[VP SET DEFAULT STYLE](#)

## G

### VP Get active cell

**VP Get active cell** ( *vpAreaName* : Text { ; *sheet* : Integer } ) : Object

Parameter	Type	Description
<i>vpAreaName</i>	Text	->4D View Pro area form object name
<i>sheet</i>	Integer	-> Sheet index (current sheet if omitted)
Result	Object	<-Range object of single cell

### Description

The `VP Get active cell` command returns a new range object referencing the cell which has the focus and where new data will be entered (the active cell).

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

In the optional *sheet* parameter, you can designate a specific spreadsheet where the range will be defined (counting begins at 0). If omitted or if you pass `vk current sheet`, the current spreadsheet is used.

## Example

The following code will retrieve the coordinates of the active cell:

```
$activeCell:=VP Get active cell("myVPArea")

//returns a range object containing:
//$activeCell.ranges[0].column=3
//$activeCell.ranges[0].row=4
//$activeCell.ranges[0].sheet=0
```

## See also

[VP ADD SELECTION](#)  
[VP Get selection](#)  
[VP RESET SELECTION](#)  
[VP SET ACTIVE CELL](#)  
[VP SET SELECTION](#)  
[VP SHOW CELL](#)

## VP Get binding path

- History

**VP Get binding path** (*rangeObj* : Object) : Text

Parameter	Type	Description
rangeObj	Object->Range object	
Result	Text	<- Name of the attribute bound to the cell

## Description

The `VP Get binding path` command returns the name of the attribute bound to the cell specified in *rangeObj*.

In *rangeObj*, pass an object that is either a cell range or a combined range of cells.  
Note that:

- If *rangeObj* is a range with several cells, the command returns the attribute name linked to the first cell in the range.
- If *rangeObj* contains several ranges of cells, the command returns the attribute name linked to the first cell of the first range.

## Example

```
var $p; $options : Object
var $myAttribute : Text

$p:=New object
$p.firstName:="Freehafer"
$p.lastName:="Nancy"

VP SET DATA CONTEXT("ViewProArea"; $p)

VP SET BINDING PATH(VP Cell("ViewProArea"; 0; 0); "firstName")
VP SET BINDING PATH(VP Cell("ViewProArea"; 1; 0); "lastName")

$myAttribute:=VP Get binding path(VP Cell("ViewProArea"; 1; 0)) // "lastName"
```

## See also

[VP SET BINDING PATH](#)

[VP Get data context](#)

[VP SET DATA CONTEXT](#)

## VP Get cell style

**VP Get cell style ( rangeObj : Object ) : Object**

### Parameter Type Description

rangeObj Object->Range object

Result Object <-Style object

### Description

The `VP Get cell style` command returns a [style object](#) for the first cell in the *rangeObj*.

In *rangeObj*, pass a range containing the style to retrieve.

- If *rangeObj* contains a cell range, the cell style is returned.
- If *rangeObj* contains a range that is not a cell range, the style of the first cell in the range is returned.
- If *rangeObj* contains several ranges, only the style of the first cell in the first range is returned.

### Example

To get the details about the style in the selected cell (B2):

	A	B	C
1			
2		H e l l o	
3		W o r l d	

This code:

```
$cellStyle:=VP Get cell style(VP Get selection("myDoc"))
```

... will return this object:

```
{
  "backColor": "Azure",
  "borderBottom": {
    "color": "#800080",
    "style": 5
  },
  "font": "8pt Arial",
  "foreColor": "red",
  "hAlign": 1,
  "isVerticalText": "true",
  "vAlign": 0
}
```

## See also

[VP GET DEFAULT STYLE](#)

[VP SET CELL STYLE](#)

## VP Get column attributes

**VP Get column attributes** (*rangeObj* : Object) : Collection

Parameter	Type	Description
-----------	------	-------------

rangeObj Object -> Range object

Result Collection <- Collection of column properties

## Description

The `VP Get column attributes` command returns a collection of properties for any column in the *rangeObj*.

In *rangeObj*, pass an object containing a range of the columns whose attributes will be retrieved.

The returned collection contains any properties for the columns, whether or not they have been set by the [VP SET COLUMN ATTRIBUTES](#) command.

## Example

The following code:

```
C_OBJECT($range)
C_COLLECTION($attr)

$range:=VP Column("ViewProArea";1;2)
$attr:=VP Get column attributes($range)
```

... will return a collection of the attributes within the given range:

## See also

[VP Get row attributes](#)

[VP SET COLUMN ATTRIBUTES](#)

[VP SET ROW ATTRIBUTES](#)

## VP Get column count

**VP Get column count** (*vpAreaName* : Text { ; *sheet* : Integer } ) : Integer

Parameter	Type	Description
-----------	------	-------------

*vpAreaName* Text -> 4D View Pro area from object name

*sheet* Integer -> Sheet index (current sheet if omitted)

Result Integer <- Total number of columns

## Description

The `VP Get column count` command returns the total number of columns from the designated *sheet*.

In `vpAreaName`, pass the name property of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

You can define where to get the column count in the optional `sheet` parameter using the sheet index (counting begins at 0). If omitted or if you pass `vk current sheet`, the current spreadsheet is used.

## Example

The following code returns the number of columns in the 4D View Pro area:

```
C_Integer($colCount)
$colCount:=VP Get column count("ViewProarea")
```

## See also

[VP Get row count](#)  
[VP SET COLUMN COUNT](#)  
[VP SET ROW COUNT](#)

## VP Get current sheet

**VP Get current sheet** ( `vpAreaName` : Text )

Parameter	Type	Description
<code>vpAreaName</code>	Text	-> 4D View Pro area form object name
Function resultInteger<-Index of the current sheet		

## Description

The `VP Get current sheet` command returns the index of the current sheet in `vpAreaName`. The current sheet is the selected sheet in the document.

In `vpAreaName`, pass the name of the 4D View Pro area.

Indexing starts at 0.

## Example

When the third sheet is selected:



The command returns 2:

```
$index:=VP Get current sheet("ViewProArea")
```

## See also

[VP SET CURRENT SHEET](#)

## VP Get data context

- History

**VP Get data context** ( `vpAreaName` : Text {; `sheet` : Integer } ) : Object  
**VP Get data context** ( `vpAreaName` : Text {; `sheet` : Integer } ) : Collection

Parameter	Type	Description
vpAreaName	Object	-> 4D View Pro area form object name
sheet	Integer	-> Index of the sheet to get the data context from
Result	Object   Collection	<- Data context

## Description

The `VP Get data context` command returns the current data context of a worksheet. The returned context includes any modifications made to the contents of the data context.

In *sheet*, pass the index of the sheet to get the data context from. If no index is passed, the command returns the data context of the current worksheet. If there is no context for the worksheet, the command returns `Null`.

The function returns an object or a collection depending on the type of data context set with [VP SET DATA CONTEXT](#).

## Example

To get the data context bound to the following cells:

	0	1	2	3
1	Freehafer	Nancy		
2				
3				

```
var $dataContext : Object
$dataContext:=VP Get data context("ViewProArea") // 
{firstName:Freehafer,lastName:Nancy}
```

## See also

[VP SET DATA CONTEXT](#)

[VP Get binding path](#)

[VP SET BINDING PATH](#)

## VP Get default style

**VP Get default style** ( *vpAreaName* : Text { ; *sheet* : Integer } ) : Integer

Parameter	Type	Description
vpAreaName	Text	-> 4D View Pro area from object name
sheet	Integer	-> Sheet index (current sheet if omitted)
Result	Integer	<- Total number of columns

## Description

The `VP Get default style` command returns a default style object for a sheet. The returned object contains basic document rendering properties as well as the default style settings (if any) previously set by the [VP SET DEFAULT STYLE](#) method. For more information about style properties, see [Style Objects & Style Sheets](#).

In *vpAreaName*, pass the name property of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

You can define where to get the column count in the optional `sheet` parameter using the sheet index (counting begins at 0). If omitted or if you pass `vk current sheet`, the current spreadsheet is used.

## Example

To get the details about the default style for this document:

	A	B	C
1			
2		Hello World!	
3			
4			
5			
6			
7			
8			

This code:

```
$defaultStyle:=VP Get default style("myDoc")
```

will return this information in the `$defaultStyle` object:

```
{
  backColor:#E6E6FA,
  hAlign:0,
  vAlign:0,
  font:12pt papyrus
}
```

## See also

[VP Get cell style](#)

[VP SET DEFAULT STYLE](#)

## VP Get formula

**VP Get formula** (`rangeObj` : Object) : Text

### Parameter Type Description

rangeObj Object->Range object  
Result Text <-Formula

### Description

The `VP Get formula` command retrieves the formula from a designated cell range.

In `rangeObj`, pass a range whose formula you want to retrieve. If `rangeObj` designates multiple cells or multiple ranges, the formula of the first cell is returned. If `rangeObj` is a cell that does not contain a formula, the method returns an empty string.

## Example

```
//set a formula
VP SET FORMULA(VP Cell("ViewProArea";5;2);"SUM($A$1:$C$10)")

$result:=VP Get formula(VP Cell("ViewProArea";5;2)) //
$result="SUM($A$1:$C$10)"
```

## See also

[VP Get formulas](#)

[VP SET FORMULA](#)

[VP SET ROW COUNT](#)

## VP Get formula by name

**VP Get formula by name** ( *vpAreaName* : Text ; *name* : Text { ; *scope* : Number } )  
: Object

Parameter	Type	Description
<i>vpAreaName</i>	Text	->4D View Pro area form object name
<i>name</i>	Text	->Name of the named range
<i>scope</i>	Number	->Target scope (default=current sheet)
Result	Text	<-Named formula or named range definition

## Description

The `VP Get formula by name` command returns the formula and comment corresponding to the named range or named formula passed in the *name* parameter, or **null** if it does not exist in the defined scope.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

Pass the named range or named formula that you want to get in *name*. Note that named ranges are returned as formulas containing absolute cell references.

You can define where to get the formula in *scope* using either the sheet index (counting begins at 0) or the following constants:

- `vk current sheet`
- `vk workbook`

## Returned Object

The returned object contains the following properties:

Property	Type	Description
<i>formula</i>	Text	Text of the formula corresponding to the named formula or named formula Text range. For named ranges, the formula is a sequence of absolute coordinates.
<i>comment</i>	Text	Comment corresponding to the named formula or named range

## Example

```
$range:=VP Cell("ViewProArea";0;0)
VP ADD RANGE NAME("Total1";$range)

$formula:=VP Get formula by name("ViewProArea";"Total1")
//$formula.formula=Sheet1!$A$1

$formula:=VP Get formula by name("ViewProArea";"Total")
//$formula=null (if not existing)
```

## See also

## [VP ADD FORMULA NAME](#)

## [VP Get names](#)

## **VP Get formulas**

**VP Get formulas** ( *rangeObj* : Object ) : Collection

Parameter	Type	Description
<i>rangeObj</i>	Object	-> Range object
Result	Collection	<- Collection of formula values

### **Description**

The `VP Get formulas` command retrieves the formulas from a designated *rangeObj*.

In *rangeObj*, pass a range whose formulas you want to retrieve. If *rangeObj* designates multiple ranges, the formula of the first range is returned. If *rangeObj* does not contain any formulas, the command returns an empty string.

The returned collection is two-dimensional:

- The first-level collection contains subcollections of formulas. Each subcollection represents a row.
- Each subcollection defines cell values for the row. Values are text elements containing the cell formulas.

### **Example**

You want to retrieve the formulas in the Sum and Average columns from this document:

You can use this code:

```
$formulas:=VP Get formulas(VP Cells("ViewProArea";5;1;2;3))
//$formulas[0]=[Sum(B2:D2),Average(B2:D2)]
//$formulas[1]=[Sum(B3:D3),Average(B3:D3)]
//$formulas[2]=[Sum(B4:D4),Average(C4:D4)]
```

### **See also**

[VP Get formula](#)  
[VP Get values](#)  
[VP SET FORMULAS](#)  
[VP SET VALUES](#)

## **VP Get frozen panes**

**VP Get frozen panes** ( *vpAreaName* : Text { ; *sheet* : Integer } ) : Object

Parameter	Type	Description
<i>vpAreaName</i>	Text	-> 4D View Pro area form object name
<i>sheet</i>	Integer	-> Sheet index (current sheet if omitted)
Result	Object	<- Object containing frozen column and row information

### **Description**

The `VP Get frozen panes` command returns an object with information about the frozen columns and rows in `vpAreaName`.

In `vpAreaName`, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

In the optional `sheet` parameter, you can designate a specific spreadsheet where the range will be defined (counting begins at 0). If omitted or if you pass `vk current sheet`, the current spreadsheet is used.

## Returned object

The command returns an object describing the frozen columns and rows. This object can contain the following properties:

Property	Type	Description
<code>columnCount</code>	<code>Integer</code>	The number of frozen columns on the left of the sheet
<code>trailingColumnCount</code>	<code>Integer</code>	The number of frozen columns on the right of the sheet
<code>rowCount</code>	<code>Integer</code>	The number of frozen rows on the top of the sheet
<code>trailingRowCount</code>	<code>Integer</code>	The number of frozen rows on the bottom of the sheet

## Example

You want to retrieve information about the number of frozen columns and rows:

```
var $panesObj : Object  
$panesObj:=VP Get frozen panes("ViewProArea")
```

The returned object contains, for example:

## See also

[VP SET FROZEN PANES](#)

## VP Get names

**VP Get names** ( `vpAreaName` : Text { ; `scope` : Number } ) : Collection

Parameter	Type	Description
<code>vpAreaName</code>	Text	->4D View Pro area form object name
<code>scope</code>	Number	-> Target scope (default= current sheet)
<code>Result</code>	Collection	<- Existing names in the defined scope

## Description

The `VP Get names` command returns a collection of all defined "names" in the current sheet or in the scope designated by the `scope` parameter.

In `vpAreaName`, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

You can define where to get the names in `scope` using either the sheet index

(counting begins at 0) or the following constants:

- `vk current sheet`
- `vk workbook`

## Returned collection

The returned collection contains one object per name. The following object properties can be returned:

Property	Type	Description
<code>result[ ].name</code>	Text	cell or range name
<code>result[ ].formula</code>	Text	formula
<code>result[ ].commentText</code>	Comment	associated to the name

Available properties depend on the type of the named element (named cell, named range, or named formula).

## Example

```
var $list : Collection  
  
$list:=VP Get names ("ViewProArea";2) //names in 3rd sheet
```

## See also

[VP ADD FORMULA NAME](#)  
[VP ADD RANGE NAME](#)  
[VP Get formula by name](#)  
[VP Name](#)

## VP Get print info

**VP Get print info** ( `vpAreaName` : Text { ; `sheet` : Integer } ) : Object

Parameter	Type	Description
<code>vpAreaName</code>	Text	->4D View Pro area form object name
<code>sheet</code>	Integer	-> Sheet index (current sheet if omitted)
Result	Object	<-Object of printing information

## Description

The `VP Get print info` command returns an object containing the print attributes of the `vpAreaName`.

Pass the the name of the 4D View Pro area in `vpAreaName`. If you pass a name that does not exist, an error is returned.

In the optional `sheet` parameter, you can designate a specific spreadsheet (counting begins at 0) whose printing attributes you want returned. If omitted or if you pass `vk current sheet`, the current spreadsheet is used.

## Example

This code:

```
$pinfo:=VP Get print info("ViewProArea")
```

... returns the print attributes of the 4D View Pro area set in the [VP SET PRINT INFO](#) command:

```
{
bestFitColumns:false,
bestFitRows:false,
blackAndWhite:false,
centering:0,
columnEnd:8,
columnStart:0,
firstPageNumber:1,
fitPagesTall:1,
fitPagesWide:1,
footerCenter:"&BS.H.I.E.L.D. &A Sales Per Region",
footerCenterImage:,  
footerLeft:,  
footerLeftImage:,  
footerRight:"page &P of &N",  
footerRightImage:,  
headerCenter:,  
headerCenterImage:,  
headerLeft:&G",  
headerLeftImage:logo.png,  
headerRight:,  
headerRightImage:,  
margin:{top:75,bottom:75,left:70,right:70,header:30,footer:30},  
orientation:2,  
pageOrder:0,  
pageRange:,  
paperSize:{width:850,height:1100,kind:1},  
qualityFactor:2,  
repeatColumnEnd:-1,  
repeatColumnStart:-1,  
repeatRowEnd:-1,  
repeatRowStart:-1,  
rowEnd:24,  
rowStart:0,  
showBorder:false,  
showColumnHeader:0,  
showGridLine:false,  
showRowHeader:0,  
useMax:true,  
watermark:[],  
zoomFactor:1
}
```

## See also

[4D View Pro Print Attributes](#)

[VP SET PRINT INFO](#)

## VP Get row attributes

**VP Get row attributes** ( rangeObj : Object ) : Collection

Parameter	Type	Description
rangeObj	Object	-> Range object
Result	Collection	<- Collection of row properties

## Description

The `VP Get row attributes` command returns a collection of properties for any row in the *rangeObj*.

In *rangeObj*, pass an object containing a range of the rows whose attributes will be retrieved.

The returned collection contains any properties for the rows, whether or not they have been set by the [VP SET ROW ATTRIBUTES](#) method.

## Example

The following code returns a collection of the attributes within the given range:

```
var $range : Object
var $attr : Collection

$range:=VP Column ("ViewProArea";1;2)
$attr:=VP Get row attributes ($range)
```

## See also

[VP Get column attributes](#)  
[VP SET COLUMN ATTRIBUTES](#)  
[VP SET ROW ATTRIBUTES](#)

## VP Get row count

**VP Get row count** ( *vpAreaName* : Text {; *sheet* : Integer} ) : Integer

Parameter	Type	Description
<i>vpAreaName</i>	Text ->	4D View Pro area from object name
<i>sheet</i>	Integer->	Sheet index (current sheet if omitted)
Result	Integer<-	Total number of rows

## Description

The `VP Get row count` command returns the total number of rows from the designated *sheet*.

In *vpAreaName*, pass the name property of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

You can define where to get the row count in the optional *sheet* parameter using the sheet index (counting begins at 0). If omitted or if you pass `vk current sheet`, the current spreadsheet is used.

## Example

The following code returns the number of rows in the 4D View Pro area:

```
var $rowCount : Integer
$rowCount:=VP Get row count ("ViewProarea")
```

## See also

[VP Get column count](#)  
[VP SET COLUMN COUNT](#)  
[VP SET ROW COUNT](#)

## VP Get selection

**VP Get selection** ( *vpAreaName* : Text {; *sheet* : Integer} ) : Object

Parameter	Type	Description
<i>vpAreaName</i>	Text	-> 4D View Pro area from object name
<i>sheet</i>	Integer	-> Sheet index (current sheet if omitted)
Result	Object	<- Range object of cells

### Description

The `VP Get selection` command returns a new range object referencing the current selected cells.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

In the optional *sheet* parameter, you can designate a specific spreadsheet where the range will be defined (counting begins at 0). If omitted or if you pass `vk current sheet`, the current spreadsheet is used.

### Example

The following code will retrieve the coordinates of all the cells in the current selection:

```
$currentSelection:=VP Get selection("myVPArea")  
  
//returns a range object containing:  
//$currentSelection.ranges[0].column=5  
//$currentSelection.ranges[0].columnCount=2  
//$currentSelection.ranges[0].row=8  
//$currentSelection.ranges[0].rowCount=6
```

### See also

[VP ADD SELECTION](#)  
[VP Get active cell](#)  
[VP SET ACTIVE CELL](#)  
[VP SET SELECTION](#)  
[VP SHOW CELL](#)

## VP Get sheet count

**VP Get sheet count** ( *vpAreaName* : Text ) : Integer

Parameter	Type	Description
<i>vpAreaName</i>	Text	-> 4D View Pro area form object name
Function result		Integer <- Number of sheets

### Description

The `VP Get sheet count` command returns the number of sheets in the document loaded in `vpAreaName`.

In `vpAreaName`, pass the name of the 4D View Pro area.

## Example

In the following document:



Get the sheet count and set the current sheet to the last sheet:

```
$count:=VP Get sheet count("ViewProArea")
//set the current sheet to the last sheet (indexing starts at 0)
VP SET CURRENT SHEET("ViewProArea";$count-1)
```



## See also

[VP Get sheet index](#)

[VP SET SHEET COUNT](#)

## VP Get sheet index

**VP Get sheet index** (`vpAreaName` : Text ; `name` : Text) : Integer

Parameter	Type	Description
<code>vpAreaName</code>	Text	-> 4D View Pro area form object name
<code>name</code>	Text	-> Sheet name
Function result		Integer <- Sheet index

## Description

The `VP Get sheet index` command returns the index of a sheet based on its name in `vpAreaName`.

In `vpAreaName`, pass the name of the 4D View Pro area.

In `name`, pass the name of the sheet whose index will be returned. If no sheet named `name` is found in the document, the method returns -1.

Indexing starts at 0.

## Example

In the following document:



Get the index of the sheet called "Total first quarter":

```
$index:=VP Get sheet index("ViewProArea";"Total first quarter")
//returns 2
```

## See also

[VP Get sheet count](#)  
[VP Get sheet name](#)

### VP Get sheet name

**VP Get sheet name** ( *vpAreaName* : Text ; *sheet* : Integer ) : Text

Parameter	Type	Description
<i>vpAreaName</i>	Text	-> 4D View Pro area form object name
<i>sheet</i>	Integer->	Sheet index
Function resultText	<-	Sheet name

## Description

The `VP Get sheet name` command returns the name of a sheet based on its index in *vpAreaName*.

In *vpAreaName*, pass the name of the 4D View Pro area.

In *sheet*, pass the index of the sheet whose name will be returned.

If the passed sheet index does not exist, the method returns an empty name.

Indexing starts at 0.

## Example

Get the name of the third sheet in the document:

```
$sheetName:=VP Get sheet name("ViewProArea";2)
```

## See also

[VP Get sheet index](#)

### VP Get sheet options

**VP Get sheet options** ( *vpAreaName* : Text {; *sheet* : Integer } ) : Object

Parameter	Type	Description
<i>vpAreaName</i>	Text	-> 4D View Pro area from object name
<i>sheet</i>	Integer->	Sheet index (current sheet if omitted)
Result	Object	<- Sheet options object

## Description

The `VP Get sheet options` command returns an object containing the current sheet options of the *vpAreaName* area.

Pass the name of the 4D View Pro area in *vpAreaName*. If you pass a name that does not exist, an error is returned.

In the optional *sheet* parameter, you can designate a specific spreadsheet (counting begins at 0). If omitted or if you pass `vk current sheet`, the current spreadsheet is used.

## Returned object

The method returns an object containing the current values for all available sheet options. An option value may have been modified by the user or by the [VP SET SHEET OPTIONS](#) method.

To view the full list of the options, see [Sheet Options](#).

## Example

```
$options:=VP Get sheet options("ViewProArea")
If($options.colHeaderVisible) //column headers are visible
    ... //do something
End if
```

## See also

[4D VIEW PRO SHEET OPTIONS](#)

[VP SET SHEET OPTIONS](#)

## VP Get show print lines

**VP Get show print lines** ( *vpAreaName* : Text {; *sheet* : Integer} ) : Boolean

Parameter	Type	Description
<i>vpAreaName</i>	Text	->4D View Pro area form object name
<i>sheet</i>	Integer	<-Sheet index
Function result	Boolean	<- True if print lines are visible, False otherwise

## Description

The `VP Get show print lines` command returns `True` if the print preview lines are visible and `False` if they are hidden.

In *vpAreaName*, pass the name of the 4D View Pro area.

In *sheet*, pass the index of the target sheet. If *sheet* is omitted, the command applies to the current sheet.

Indexing starts at 0.

## Example

The following code checks if preview lines are displayed or hidden in the document:

```
var $result : Boolean
$result:=VP Get show print lines("ViewProArea";1)
```

## See also

[VP SET SHOW PRINT LINES](#)

## VP Get spans

## **VP Get spans ( *rangeObj* : Object ) : Object**

Parameter	Type	Description
rangeObj	Object->Range object	
Result	Object<- range	Object of cell spans in the defined range

### **Description**

The `VP Get spans` command retrieves the cell spans in the designated *rangeObj*.

In *rangeObj*, pass a range of cell spans you want to retrieve. If *rangeObj* does not contain a cell span, an empty range is returned.

### **Example**

You want to center the text for the spanned cells in this document:

```
// Search for all cell spans
$range:=VP Get spans(VP All("ViewProArea"))

//center text
$style:=New object("vAlign";vk vertical align center;"hAlign";vk
horizontal align center)
VP SET CELL STYLE($range;$style)
```

### **See also**

[VP ADD SPAN](#)

[VP REMOVE SPAN](#)

## **VP Get stylesheet**

### **VP Get stylesheet ( *vpAreaName* : Text ; *styleName* : Text { ; *sheet* : Integer } ) : Object**

Parameter	Type	Description
vpAreaName	Text	->4D View Pro area form object name
styleName	Text	->Name of style
sheet	Integer->	Sheet index (current sheet if omitted)

|Result|Object|<-|Style sheet object|

### **Description**

The `VP Get stylesheet` command returns the *styleName* style sheet object containing the property values which have been defined.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

In *styleName*, pass the name of the style sheet to get.

You can define where to get the style sheet in the optional *sheet* parameter using the sheet index (counting begins at 0) or with the following constants:

- `vk current sheet`
- `vk workbook`

## Example

The following code:

```
$style:=VP Get stylesheet("ViewProArea";"GreenDashDotStyle")
```

... will return the *GreenDashDotStyle* style object from the current sheet:

```
{
backColor:green,
borderBottom:{color:green,style:10},
borderLeft:{color:green,style:10},
borderRight:{color:green,style:10},
borderTop:{color:green,style:10}
}
```

## See also

[4D View Pro Style Objects and Style Sheets](#)

[VP ADD STYLESHEET](#)

[VP Get stylesheets](#)

[VP REMOVE STYLESHEET](#)

## VP Get stylesheets

**VP Get stylesheets** (*vpAreaName* : Text { ; *sheet* : Integer } ) : Collection

Parameter	Type	Description
<i>vpAreaName</i>	Text	-> 4D View Pro area form object name
<i>sheet</i>	Integer	-> Target scope (default = current sheet)
Result	Collection	<- Collection of style sheet objects

## Description

The `VP Get stylesheets` command returns the collection of defined style sheet objects from the designated *sheet*.

In *vpAreaName*, pass the name property of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

You can define where to get the style sheets in the optional *sheet* parameter using the sheet index (counting begins at 0) or with the following constants:

- `vk current sheet`
- `vk workbook`

## Example

The following code will return a collection of all the style objects in the current sheet:

```
$styles:=VP Get stylesheets("ViewProArea")
```

In this case, the current sheet uses two style objects:

```

[
  {
    backColor:green,
    borderLeft:{color:green,style:10},
    borderTop:{color:green,style:10},
    borderRight:{color:green,style:10},
    borderBottom:{color:green,style:10},
    name:GreenDashDotStyle
  },
  {
    backColor:red,
    textIndent:10,
    name:RedIndent
  }
]

```

## See also

[VP ADD STYLESHEET](#)

[VP Get stylesheet](#)

[VP REMOVE STYLESHEET](#)

## VP Get table column attributes

- History

**VP Get table column attributes** ( *vpAreaName* : Text ; *tableName* : Text ; *column* : Integer {; *sheet* : Integer } ) : Object

Parameter	Type	Description
<i>vpAreaName</i>	Text	->4D View Pro area form object name
<i>tableName</i>	Text	->Table name
<i>column</i>	Integer	->Index of the column in the table
<i>sheet</i>	Integer	-> Sheet index (current sheet if omitted)
Result	Object	<-Attributes of the <i>column</i>

### Description

The `VP Get table column attributes` command returns the current attributes of the specified *column* in the *tableName*.

In *vpAreaName*, pass the name of the 4D View Pro area.

In *sheet*, pass the index of the target sheet. If no index is specified or if you pass -1, the command applies to the current sheet.

Indexing starts at 0.

The command returns an object describing the current attributes of the *column*:

Property	Type	Description
<i>dataField</i>	text	Table column's property name in the data context. Not returned if the table is displayed automatically
<i>name</i>	text	Table column's name.
<i>footerText</i>	text	Column footer value.
<i>footerFormula</i>	text	Column footer formula.
<i>filterButtonVisible</i>	boolean	True if the table column's filter button is displayed, False otherwise.

If *tableName* is not found or if *column* index is higher than the number of columns, the command returns **null**.

## Example

```
var $attributes : Object
$attributes:=VP Get table column attributes("ViewProArea"; $tableName;
1)
If ($attributes.dataField#"")
    ...
End if
```

## See also

[VP CREATE TABLE](#)  
[VP Find table](#)  
[VP SET TABLE COLUMN ATTRIBUTES](#)  
[VP RESIZE TABLE](#)

## VP Get table column index

- History

**VP Get table column index** ( *vpAreaName* : Text ; *tableName* : Text ; *columnName* : Text {; *sheet* : Integer } ) : Integer

Parameter	Type	Description
<i>vpAreaName</i>	Text	->4D View Pro area form object name
<i>tableName</i>	Text	->Table name
<i>columnName</i>	Text	->Name of the table column
<i>sheet</i>	Integer->	Sheet index (current sheet if omitted)
Result	Integer<-	Index of <i>columnName</i>

## Description

The `VP Get table column index` command returns the index of the *columnName* in the *tableName*.

In *vpAreaName*, pass the name of the 4D View Pro area.

In *columnName*, pass the name of the table column for which you want to get the index.

In *sheet*, pass the index of the target sheet. If no index is specified or if you pass -1, the command applies to the current sheet.

Indexing starts at 0.

If *tableName* or *columnName* is not found, the command returns -1.

## Example

```
// Search the column id according the column name
var $id : Integer
$id:=VP Get table column index($area; $tableName; "Weight price")
    // Remove the column by id
VP REMOVE TABLE COLUMNS($area; $tableName; $id)
```

## See also

[VP CREATE TABLE](#)  
[VP Find table](#)  
[VP Get table column attributes](#)  
[VP SET TABLE COLUMN ATTRIBUTES](#)

## VP Get table dirty rows

- History

**VP Get table dirty rows** ( *vpAreaName* : Text ; *tableName* : Text { ; *reset* : Boolean {; *sheet* : Integer } } ) : Collection

Parameter	Type	Description
<i>vpAreaName</i>	Text	->4D View Pro area form object name
<i>tableName</i>	Text	->Table name
<i>reset</i>	Boolean	-> True to clear the dirty status from the current table, False to keep it untouched. Default=True
<i>sheet</i>	Integer	->Sheet index (current sheet if omitted)
<i>Result</i>	Collection<-	Collection of objects with all the items modified since the last reset

## Description

The `VP Get table dirty rows` command returns a collection of *dirty row* objects, containing items that were modified since the last reset in the specified *tableName*.

In *vpAreaName*, pass the name of the 4D View Pro area.

In *tableName*, pass the name of the table for which you want to get the dirty rows. Only modified columns bound to a [data context](#) will be taken into account.

By default, calling the command will clear the *dirty* status from the current table. To keep this status untouched, pass `False` in the *reset* parameter.

In *sheet*, pass the index of the target sheet. If no index is specified or if you pass -1, the command applies to the current sheet.

Indexing starts at 0.

Each *dirty row* object in the returned collection contains the following properties:

Property	Type	Description
<i>item</i>	object	Modified object of the modified row
<i>originalItem</i>	Object before modification	
<i>row</i>	integer	Index of the modified row

If *tableName* is not found or if it does not contain a modified column, the command returns an empty collection.

## Example

You want to count the number of edited rows:

```
var $dirty : Collection
$dirty:=VP Get table dirty rows("ViewProArea"; "ContextTable"; False)
VP SET NUM VALUE(VP Cell("ViewProArea"; 0; 0); $dirty.length)
```

## See also

[VP CREATE TABLE](#)  
[VP Find table](#)  
[VP SET TABLE COLUMN ATTRIBUTES](#)  
[VP RESIZE TABLE](#)

## VP Get table range

- History

**VP Get table range** ( *vpAreaName* : Text ; *tableName* : Text {; *onlyData* : Integer {; *sheet* : Integer } } ) : Object

Parameter	Type	Description
<i>vpAreaName</i>	Text	->4D View Pro area form object name
<i>tableName</i>	Text	->Table name
<i>onlyData</i>	Integer-> <code>vk table full range</code> (default) or <code>vk table data range</code>	
<i>sheet</i>	Integer->Sheet index (current sheet if omitted)	
Result	Object	<-Range that contains the table

## Description

The `VP Get table range` command returns the range of *tableName*.

In *vpAreaName*, pass the name of the 4D View Pro area.

In the *onlyData* parameter, you can pass one of the following constants to indicate if you want to get the data only:

Constant	Value	Description
<code>vk table full range</code>	0	Get the cell range for the table area with footer and header (default if omitted)
<code>vk table data range</code>	1	Get the cell range for the table data area only

In *sheet*, pass the index of the target sheet. If no index is specified, the command applies to the current sheet.

Indexing starts at 0.

If *tableName* is not found, the command returns **null**.

## See also

[VP RESIZE TABLE](#)  
[VP Find table](#)

## VP Get table theme

- History

**VP Get table theme** ( *vpAreaName* : Text ; *tableName* : Text ) : cs.ViewPro.TableTheme

Parameter	Type	Description
vpAreaName	Text	->4D View Pro area form object name
tableName	Text	->Table name
Result	<a href="#">cs.ViewPro.TableTheme</a>	<-Current table theme property values

## Description

The `VP Get table theme` command returns the current theme propertie values of the *tableName*. A table theme can be set using the [VP CREATE TABLE](#) or [VP SET TABLE THEME](#) commands, or through the interface.

In *vpAreaName*, pass the name of the 4D View Pro area and in *tableName*, the name of the table.

The command returns an object of the [cs.ViewPro.TableTheme](#) class with properties and values that describe the current table theme.

## Example

The command returns a full `theme` object even if a [native SpreadJS theme](#) name was used to define the theme.

```
var $param : cs.ViewPro.TableTheme
$param:=cs.ViewPro.TableTheme.new()
$param.theme:="dark10" //use of a native theme name

VP SET TABLE THEME("ViewProArea"; "ContextTable"; $param)
$vTheme:=VP Get table theme("ViewProArea"; "ContextTable")
$result:=Asserted(Value type($vTheme.theme)=Is object) //true
```

## See also

[VP CREATE TABLE](#)  
[VP SET TABLE THEME](#)

## VP Get tables

- History

**VP Get tables** (*vpAreaName* : Text { ; *sheet* : Integer } ) : Collection

Parameter	Type	Description
vpAreaName	Text	->4D View Pro area form object name
<i>sheet</i>	Integer	-> Sheet index (current sheet if omitted)
Result	Collection	<-Text collection with all table names

## Description

The `VP Get tables` command returns a collection of all table names defined in the *sheet*.

In *vpAreaName*, pass the name of the 4D View Pro area.

In *sheet*, pass the index of the target sheet. If no index is specified, the command applies to the current sheet.

Indexing starts at 0.

## Example

The following code will return a collection of all the table names in the current sheet:

```
$tables:=VP Get tables("ViewProArea")
//$tables contains for example ["contextTable","emailTable"]
```

## See also

[VP CREATE TABLE](#)

## VP Get value

**VP Get value** ( *rangeObj* : Object ) : Object

Parameter	Type	Description
<i>rangeObj</i>	Object-> Range object	
Result	Object <-	Object containing a cell value

## Description

The `VP Get value` command retrieves a cell value from a designated cell range.

In *rangeObj*, pass a range whose value you want to retrieve.

## Returned object

The object returned will contain the `value` property, and, in case of a js date value, a `time` property:

Property	Type	Description
<code>value</code>	Integer, Real, Boolean, Text, Date	Value in the <i>rangeObj</i> (except- time)
<code>time</code>	Real	Time value (in seconds) if the value is of the js date type

If the object returned includes a date or time, it is treated as a datetime and completed as follows:

- time value - the date portion is completed as December 30, 1899 in dd/MM/yyyy format (30/12/1899)
- date value - the time portion is completed as midnight in HH:mm:ss format (00:00:00)

If *rangeObj* contains multiple cells or multiple ranges, the value of the first cell is returned. The command returns a null object if the cell is empty.

## Example

```
$cell:=VP Cell("ViewProArea";5;2)
$value:=VP Get value($cell)
If(Value type($value.value)=Is text)
    VP SET TEXT VALUE($cell;New
object("value";Uppercase($value.value)))
End if
```

## See also

[VP Get values](#)  
[VP SET VALUE](#)  
[VP SET VALUES](#)

## VP Get values

**VP Get values** ( *rangeObj* : Object ) : Collection

Parameter	Type	Description
<i>rangeObj</i>	Object	-> Range object
Result	Collection	<- Collection of values

### Description

The `VP Get values` command retrieves the values from the designated *rangeObj*.

In *rangeObj*, pass a range whose values you want to retrieve. If *rangeObj* includes multiple ranges, only the first range is used.

The collection returned by `VP Get values` contains a two-dimensional collection:

- Each element of the first-level collection represents a row and contains a subcollection of values
- Each subcollection contains cell values for the row. Values can be Integer, Real, Boolean, Text, Null. If a value is a date or time, it is returned in an object with the following properties:

Property	Type	Description
value	Date	Value in the cell (except- time)
time	Real	Time value (in seconds) if the value is of the js date type

Dates or times are treated as a datetime and completed as follows:

- time value - the date portion is completed as December 30, 1899
- date value - the time portion is completed as midnight (00:00:00:000)

### Example

You want to get values from C4 to G6:

```
$result:=VP Get values(VP Cells("ViewProArea";2;3;5;3))  
// $result[0]=[4,5,null,hello,world]  
// $result[1]=[6,7,8,9,null]  
// $result[2]=[null,{time:42,value:2019-05-  
29T00:00:00.000Z},null,null,null]
```

### See also

[VP Get formulas](#)  
[VP Get value](#)  
[VP SET FORMULAS](#)  
[VP SET VALUES](#)

## VP Get workbook options

## **VP Get workbook options ( *vpAreaName* : Text ) : Object**

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>vpAreaName</i>	Text	-> 4D View Pro area form object name
Result	Object	<- Object containing the workbook options

### **Description**

`VP Get workbook options` returns an object containing all the workbook options in *vpAreaName*

In *vpAreaName*, pass the name of the 4D View Pro area.

The returned object contains all the workbook options (default and modified ones), in the workbook.

The list of workbook options is referenced in [VP SET WORKBOOK OPTIONS's description](#).

### **Example**

```
var $workbookOptions : Object  
$workbookOptions:=VP Get workbook options("ViewProArea")
```

### **See also**

[VP SET WORKBOOK OPTIONS](#)

## **I**

### **VP IMPORT DOCUMENT**

## **VP IMPORT DOCUMENT ( *vpAreaName* : Text ; *filePath* : Text { ; *paramObj* : Object} )**

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>vpAreaName</i>	Text	-> 4D View Pro area form object name
<i>filePath</i>	Text	-> Pathname of the document
<i>paramObj</i>	Object	-> Import options

### **Description**

The `VP IMPORT DOCUMENT` command imports and displays the document designated by *filePath* in the 4D View Pro area *vpAreaName*. The imported document replaces any data already inserted in the area.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

In *filePath*, pass the path and name of the document to be imported. The following formats are supported :

- 4D View Pro documents (extension ".4vp")
- Microsoft Excel (extension ".xlsx")

- text documents (extension ".txt", ".csv", the document must be in utf-8)

If the document extension is not a recognized extension, such as `.4vp` or `.xlsx`, the document is considered a text document. You must pass a full path, unless the document is located at the same level as the Project folder, in which case you can just pass its name.

When importing a Microsoft Excel-formatted file into a 4D View Pro document, some settings may be lost. You can verify your settings with [this list from GrapeCity](#).

An error is returned if the `filePath` parameter is invalid, or if the file is missing or malformed.

The optional `paramObj` parameter allows you to define properties for the imported document:

Parameter	Type	Description
formula	object	A callback method name to be launched when the import has completed. The method must use the <code>Formula</code> command. See <a href="#">Passing a callback method (formula)</a> .
password	text	Microsoft Excel only (optional) - The password used to protect a MS Excel document.
csvOptions	object	options for csv import
range	object	Cell range that contains the first cell where the data will be written. If the specified range is not a cell range, only the first cell of the range is used.
rowDelimiter	text	Row delimiter. If not present, the delimiter is automatically determined by 4D.
columnDelimiter	text	Column delimiter. Default: ","

For more information on the CSV format and delimiter-separated values in general, see [this article on Wikipedia](#)

## Example 1

You want to import a default 4D View Pro document stored on the disk when the form is open:

```
C_TEXT($docPath)
If (Form event code=On VP Ready) //4D View Pro area loaded and ready
  $docPath:="C:\\Bases\\ViewProDocs\\MyExport.4VP"
  VP IMPORT DOCUMENT("VPArea";$docPath)
End if
```

## Example 2

You want to import a password protected Microsoft Excel document into a 4D View Pro area:

```
$o:=New object
$o.password:="excel123"

VP IMPORT DOCUMENT("ViewProArea";"c:\\tmp\\excelfilefile.xlsx";$o)
```

## Example 3

You want to import a `.txt` file that uses a comma (",") as delimiter:

```
"Clark","Kent"  
"Bruce","Wayne"  
"Barry","Allen"  
"Peter","Parker"  
"Tony","Stark"
```

```
$params:=New object  
$params.range:=VP Cells("ViewProArea";0;0;2;5)  
VP IMPORT DOCUMENT("ViewProArea";"c:\\import\\my-file.txt";New  
object("csvOptions";$params))
```

Here's the result:

## See also

[VP EXPORT DOCUMENT](#)

[VP NEW DOCUMENT](#)

## VP IMPORT FROM OBJECT

**VP IMPORT FROM OBJECT** ( *vpAreaName* : Text { ; *viewPro* : Object} )

Parameter	Type	Description
<i>vpAreaName</i>	Text ->	4D View Pro area form object name
<i>viewPro</i>	Object->	4D View Pro object

## Description

The `VP IMPORT FROM OBJECT` command imports and displays the *viewPro* 4D View Pro object in the *vpAreaName* 4D View Pro area. The imported object contents replaces any data already inserted in the area.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

In *viewPro*, pass a valid 4D View Pro object. This object can have been created using [VP Export to object](#) or manually. For more information on 4D View Pro objects, please refer to the [4D View Pro object](#) section.

An error is returned if the *viewPro* object is invalid.

## Example

You want to import a spreadsheet that was previously saved in an object field:

```
QUERY ( [VPWorkBooks] ; [VPWorkBooks] ID=10)  
VP IMPORT FROM OBJECT ("ViewProArea1"; [VPWorkBooks] SPBook)
```

## See also

[VP Export to object](#)

## VP INSERT COLUMNS

## **VP INSERT COLUMNS ( *rangeObj* : Object )**

### **Parameter Type    Description**

rangeObj Object->Range object

### **Description**

The `VP INSERT COLUMNS` command inserts columns into the *rangeObj*.

In *rangeObj*, pass an object containing a range of the starting column (the column which designates where the new column will be inserted) and the number of columns to insert. If the number of column to insert is omitted (not defined), a single column is inserted.

New columns are inserted on the left, directly before the starting column in the *rangeObj*.

### **Example**

To insert three columns before the second column:

```
VP INSERT COLUMNS(VP Column("ViewProArea";1;3))
```

The results is:

### **See also**

[VP DELETE COLUMNS](#)

[VP DELETE ROWS](#)

[VP INSERT ROWS](#)

## **VP INSERT ROWS**

### **VP INSERT ROWS ( *rangeObj* : Object )**

### **Parameter Type    Description**

rangeObj Object->Range object

### **Description**

The `VP INSERT ROWS` command inserts rows defined by the *rangeObj*.

In *rangeObj*, pass an object containing a range of the starting row (the row which designates where the new row will be inserted) and the number of rows to insert. If the number of rows to insert is omitted (not defined), a single row is inserted.

New rows are inserted directly before the first row in the *rangeObj*.

### **Example**

To insert 3 rows before the first row:

```
VP INSERT ROWS(VP Row("ViewProArea";0;3))
```

The results is:

## See also

[VP DELETE COLUMNS](#)  
[VP DELETE ROWS](#)  
[VP INSERT COLUMNS](#)

## VP INSERT TABLE COLUMNS

- History

**VP INSERT TABLE COLUMNS** ( *vpAreaName* : Text ; *tableName* : Text ; *column* : Integer {; *count* : Integer {; *insertAfter* : Integer {; *sheet* : Integer }}} )

Parameter	Type	Description
<i>vpAreaName</i>	Text	-> 4D View Pro area form object name
<i>tableName</i>	Text	-> Table name
<i>column</i>	Integer	-> Index in the table of the starting column to insert
<i>count</i>	Text	-> Number of columns to add (must be >0)
<i>insertAfter</i>	Integer	-> <code>vk table insert before</code> or <code>vk table insert after</code> <i>column</i>
<i>sheet</i>	Integer	-> Sheet index (current sheet if omitted)

## Description

The `VP INSERT TABLE COLUMNS` command inserts one or *count* empty column(s) in the specified *tableName* at the specified *column* index.

When a column has been inserted with this command, you typically modify its contents using the [VP SET TABLE COLUMN ATTRIBUTES](#) command.

In the *insertAfter* parameter, you can pass one of the following constants to indicate if the column(s) must be inserted before or after the *column* index:

Constant	Value	Description
<code>vk table insert before</code>	0	Insert column(s) before the <i>column</i> (default if omitted)
<code>vk table insert after</code>	1	Insert column(s) after the <i>column</i>

This command inserts some columns in the *tableName* table, NOT in the sheet. The total number of columns of the sheet is not impacted by the command. Data present at the right of the table (if any) are automatically moved right according to the number of added columns.

If *tableName* does not exist or if there is not enough space in the sheet, nothing happens.

## Example

See examples for [VP INSERT TABLE ROWS](#) and [VP SET TABLE COLUMN ATTRIBUTES](#).

## See also

[VP INSERT TABLE ROWS](#)

[VP REMOVE TABLE COLUMNS](#)

[VP SET TABLE COLUMN ATTRIBUTES](#)

## VP INSERT TABLE ROWS

- History

**VP INSERT TABLE ROWS** ( *vpAreaName* : Text ; *tableName* : Text ; *row* : Integer {; *count* : Integer {; *insertAfter* : Integer {; *sheet* : Integer }}} )

Parameter	Type	Description
<i>vpAreaName</i>	Text	->4D View Pro area form object name
<i>tableName</i>	Text	->Table name
<i>row</i>	Integer	->Index in the table of the starting row to insert
<i>count</i>	Text	->Number of rows to add (must be >0)
<i>insertAfter</i>	Integer	-> <code>vk table insert before</code> or <code>vk table insert after</code> <i>row</i>
<i>sheet</i>	Integer	->Sheet index (current sheet if omitted)

### Description

The `VP INSERT TABLE ROWS` command inserts one or *count* empty row(s) in the specified *tableName* at the specified *row* index.

In the *insertAfter* parameter, you can pass one of the following constants to indicate if the row(s) must be inserted before or after the *row* index:

Constant	Value	Description
<code>vk table insert before</code>	0	Insert row(s) before the <i>row</i> (default if omitted)
<code>vk table insert after</code>	1	Insert row(s) after the <i>row</i>

This command inserts some rows in the *tableName* table, NOT in the sheet. The total number of rows of the sheet is not impacted by the command. Data present below the table (if any) are automatically moved down according to the number of added rows.

If the *tableName* table is bound to a [data context](#), the command inserts new, empty element(s) in the collection.

If *tableName* does not exist or if there is not enough space in the sheet, nothing happens.

### Example

You create a table with a data context:

```
var $context : Object
$context:=New object()

$context.col:=New collection
$context.col.push(New object("name"; "Smith"; "salary"; 10000))
$context.col.push(New object("name"; "Wesson"; "salary"; 50000))
$context.col.push(New object("name"; "Gross"; "salary"; 10500))

VP SET DATA CONTEXT("ViewProArea"; $context)

VP CREATE TABLE(VP Cells("ViewProArea"; 1; 1; 3; 3); "PeopleTable";
"col")
```

A	B	C	D
1			
2	name	salary	
3	Smith	10000	
4	Wesson	50000	
5	Gross	10500	
6			

You want to insert two rows and two columns in the table, you can write:

```
VP INSERT TABLE ROWS ("ViewProArea"; "PeopleTable"; 1; 2)
VP INSERT TABLE COLUMNS ("ViewProArea"; "PeopleTable"; 1; 2)
```

A	B	C	D	E
1				
2	name	Column	Column	salary
3	Smith			10000
4				
5				
6	Wesson			50000
7	Gross			10500
8				

## See also

[VP INSERT TABLE COLUMNS](#)

[VP REMOVE TABLE ROWS](#)

## M

### VP MOVE CELLS

- History

**VP MOVE CELLS** ( *originRange* : Object ; *targetRange* : Object ; *options* : Object )

Parameter	Type	Description
-----------	------	-------------

*originRange* Object -> Cell range to copy from

*targetRange* Object -> Target range for the values, formatting and formulas

*options* Object -> Additional options

#### Description

The `VP MOVE CELLS` command moves or copies the values, style and formulas from *originRange* to *targetRange*.

*originRange* and *targetRange* can refer to different View Pro areas.

In *originRange*, pass a range object containing the values, style, and formula cells to copy or move. If *originRange* is a combined range, only the first one is used.

In *targetRange*, pass the range of cells where the cell values, style, and formulas will be copied or moved.

The *options* parameter has several properties:

<b>Property</b>	<b>Type</b>	<b>Description</b>
		Determines if the values, formatting and formulas of the cells in <i>originRange</i> are removed after the command executes:
copy	Boolean	<ul style="list-style-type: none"> <li>• <i>False</i> (default) to remove them</li> <li>• <i>True</i> to keep them</li> </ul>
		Specifies what is pasted. Possible values:
	<b>Value</b>	<b>Description</b>
	vk clipboard options all (default)	Pastes all data objects, including values, formatting, and formulas.
	vk clipboard options formatting	Pastes only the formatting.
pasteOptions	Longint	Pastes only the formulas.
	vk clipboard options formulas	Pastes the formulas and formatting.
	vk clipboard options formulas and formatting	Pastes only the values.
	vk clipboard options values	Pastes the values and formatting.
	vk clipboard options value and formatting	

The paste options defined in the [workbook options](#) are taken into account.

## Example

To copy the contents, values, formatting and formulas from an origin range:

```
var $originRange; $targetRange; $options : Object
$originRange:=VP Cells("ViewProArea"; 0; 0; 2; 5)
$targetRange:=VP Cells("ViewProArea"; 4; 0; 2; 5)
$options:=New object
$options.copy:=True
$options.pasteOptions:=vk clipboard options all
VP MOVE CELLS($originRange; $targetRange; $options)
```

## See also

[VP Copy to object](#)  
[VP PASTE FROM OBJECT](#)  
[VP SET WORKBOOK OPTIONS](#)

## N

### VP Name

**VP Name** (*vpAreaName* : Text ; *rangeName* : Text { ; *sheet* : Integer } ) : Object

Parameter	Type	Description
vpAreaName	Text	-> 4D View Pro area form object name
rangeName	Text	-> Existing range name
sheet	Integer	-> Range location (current sheet if omitted)
Result	Object	<- Range object of name

## Description

The `VP Name` command returns a new range object referencing a named range.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

The *rangeName* parameter specifies an existing named cell range.

In the optional *sheet* parameter, you can designate a specific spreadsheet where *rangeName* is defined. If omitted, the current spreadsheet is used by default. You can explicitly select the current spreadsheet or the entire workbook with the following constants:

- `vk current sheet`
- `vk workbook`

## Example

You want to give a value to the "Total" named range.

```
// name the B5 cell as Total
VP ADD RANGE NAME(VP Cell("ViewProArea";1;4); "Total")
$name:=VP Name("ViewProArea"; " Total")
VP SET NUM VALUE($name;285;"$#,###.00")
```

## See also

[VP ADD RANGE NAME](#)  
[VP ALL](#)  
[VP Cell](#)  
[VP Cells](#)  
[VP Column](#)  
[VP Combine ranges](#)  
[VP Get names](#)  
[VP REMOVE NAME](#)  
[VP Row](#)

## VP NEW DOCUMENT

**VP NEW DOCUMENT** ( *vpAreaName* : Text )

Parameter	Type	Description
vpAreaName	Text	-> 4D View Pro area form object name

## Description

The `VP NEW DOCUMENT` command loads and display a new, default document in the 4D View Pro form area object *vpAreaName*. The new empty document replaces any data already inserted in the area.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

## Example

You want to display an empty document in the "myVPArea" form object:

```
VP NEW DOCUMENT ("myVPArea")
```

## See also

[VP IMPORT DOCUMENT](#)

# O

## VP Object to font

**VP Object to font** (*fontObj* : Object) : Text

Parameter	Type	Description
fontObj	Object->Font object	
Result	Text	<-Font shorthand

## Description

The `VP Object to font` command returns a font shorthand string from *fontObj*.

In *fontObj*, pass an object containing the font properties. The following properties are supported:

Property Type	Description	Possible values	Mandatory
family text	Specifies the font.	any standard or generic font family. Ex. "Arial", "Helvetica", "serif", "arial,sans-serif"	Yes
size text	Defines the size of the font. The line-height can be added to the font-size: font-size/line-height: Ex: "15pt/20pt"	a number with one of the following units: • "em", "ex", "%", "px", "cm", "mm", "in", "pt", "pc", "ch", "rem", "vh", "vw", "vmin", "vmax" or one of the following: • <code>vk font size large</code> • <code>vk font size larger</code> Yes • <code>vk font size x large</code> • <code>vk font size xx large</code> • <code>vk font size small</code> • <code>vk font size smaller</code> • <code>vk font size x small</code> • <code>vk font size xx small</code> • <code>vk font style italic</code> • <code>vk font style oblique</code> • <code>vk font variant small caps</code> • <code>vk font weight 100</code> • <code>vk font weight 200</code> • <code>vk font weight 300</code> • <code>vk font weight 400</code> • <code>vk font weight 500</code> • <code>vk font weight 600</code> • <code>vk font weight 700</code> • <code>vk font weight 800</code> • <code>vk font weight 900</code> • <code>vk font weight bold</code> • <code>vk font weight bolder</code> • <code>vk font weight lighter</code>	
style text	The style of the font.	<code>italic</code> <code>oblique</code>	No
variant text	Specifies font in small capital letters.	<code>small caps</code>	No
weight text	Defines the thickness of the font.	<code>100</code> <code>200</code> <code>300</code> <code>400</code> <code>500</code> <code>600</code> <code>700</code> <code>800</code> <code>900</code> <code>bold</code> <code>bolder</code> <code>lighter</code>	No

This object can be created with the [VP Font to object](#) command.

The returned shorthand string can be assigned to the "font" property of a cell with the [VP SET CELL STYLE](#), for example.

## Example

```

$cellStyle:=VP Get cell style($range)

$font:=VP Font to object($cellStyle.font)
$font.style:=vk font style oblique
$font.variant:=vk font variant small caps
$font.weight:=vk font weight bolder

$cellStyle.font:=VP Object to font($font)
// $cellStyle.font contains "bolder oblique small-caps 16pt arial"

```

## See also

[4D View Pro Style Objects and Style Sheets](#)

[VP Font to object](#)

[VP SET CELL STYLE](#)

[VP SET DEFAULT STYLE](#)

## P

### VP PASTE FROM OBJECT

- History

**VP PASTE FROM OBJECT** ( *rangeObj* : Object ; *dataObject* : Object {; *options* : Longint} )

Parameter	Type	Description
<i>rangeObj</i>	Object -> Cell range object	
<i>dataObject</i>	Object -> Object containing the data to be pasted	
<i>options</i>	Longint-> Specifies what is pasted	

#### Description

The `VP PASTE FROM OBJECT` command pastes the contents, style and formulas stored in *dataObject* to the *rangeObj* object.

In *rangeObj*, pass the cell range object where the values, formatting, and/or formula cells will be pasted. If *rangeObj* refers to more than one cell, only the first one is used.

In *dataObject*, pass the object that contains the cell data, formatting, and formulas to be pasted.

In the optional *options* parameter, you can specify what to paste in the cell range. Possible values:

Constant	Description
<code>vk clipboard options all</code>	Pastes all data objects, including values, formatting, and formulas.
<code>vk clipboard options formatting</code>	Pastes only the formatting.
<code>vk clipboard options formulas</code>	Pastes only the formulas.
<code>vk clipboard options formulas and formatting</code>	Pastes formulas and formatting.
<code>vk clipboard options values</code>	Pastes only values.
<code>vk clipboard options value and formatting</code>	Pastes values and formatting.

The paste options defined in the [workbook options](#) are taken into account.

If *options* refers to a paste option not present in the copied object (e.g. formulas), the command does nothing.

## Example

See example the example from [VP Copy to object](#)

## See also

[VP Copy to object](#)

[VP MOVE CELLS](#)

[VP Get workbook options](#)

[VP SET WORKBOOK OPTIONS](#)

## VP PRINT

**VP PRINT** ( *vpAreaName* : Text { ; *sheet* : Integer } )

Parameter	Type	Description
<i>vpAreaName</i>	Text	-> 4D View Pro area form object name
<i>sheet</i>	Integer	-> Sheet index (current sheet if omitted)

## Description

The `VP PRINT` command opens a print dialog window to print *vpAreaName*.

Pass the 4D View Pro area to be printed in *vpAreaName*. The command will open the system print dialog window where the printer can be specified and the page properties can be defined.

The properties defined in the print dialog window are for the printer paper, they are not the printing properties for the 4D View Pro area. Printing properties for 4D View Pro areas are defined using the [VP SET PRINT INFO](#) command. It is highly recommended that the properties for both the printer and the 4D View Pro area match, otherwise the printed document may not correspond to your expectations.

In the optional *sheet* parameter, you can designate a specific spreadsheet to print (counting begins at 0). If omitted, the current sheet is used by default. You can explicitly select the current spreadsheet or entire workbook with the following constants:

- `vk current sheet`
- `vk workbook`

- 4D View Pro areas can only be printed with the `VP PRINT` command.
- Commands from the 4D **Printing** language theme are not supported by `VP PRINT`.
- This command is intended for individual printing by the final end user. For automated print jobs, it is advised to export the 4D View Pro area as a PDF with the [VP EXPORT DOCUMENT](#) method.

## Example

The following code:

```
VP PRINT("myVPArea")
```

... will open a print dialog window:

## See also

[VP EXPORT DOCUMENT](#)

[VP SET PRINT INFO](#)

## R

### VP RECOMPUTE FORMULAS

**VP RECOMPUTE FORMULAS** ( *vpAreaName* : Text )

Parameter	Type	Description
<i>vpAreaName</i>	Text	-> 4D View Pro area form object name

#### Description

The `VP RECOMPUTE FORMULAS` command immediately evaluates all formulas in *vpAreaName*. By default, 4D automatically computes formulas when they are inserted, imported, or exported. `VP RECOMPUTE FORMULAS` allows you to force the compute at any time (e.g, in case modifications are made to the formulas or if the formulas contain calls to the database). The command launches the execution of the [VP FLUSH COMMANDS](#) command to execute any stored commands and clear the command buffer, then calculates all formulas in the workbook.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

Be sure the [VP SUSPEND COMPUTING](#) command has not been executed before using `VP RECOMPUTE FORMULAS`, otherwise the command does nothing.

#### Example

To refresh all formulas in the workbook:

```
VP RECOMPUTE FORMULAS ("ViewProArea")
```

## See also

[VP RESUME COMPUTING](#)

[VP SUSPEND COMPUTING](#)

### VP REMOVE NAME

**VP REMOVE NAME** ( *vpAreaName* : Text ; *name* : Text { ; *sheet* : Integer } )

Parameter	Type	Description
<i>vpAreaName</i>	Text	-> 4D View Pro area form object name
<i>name</i>	Text	-> Name of the named range or named formula to remove
<i>scope</i>	Integer	-> Target scope (default=current sheet)

#### Description

The `VP REMOVE NAME` command removes the named range or named formula passed in the *name* parameter in the defined *scope*.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

Pass the named range or named formula that you want to remove in *name*.

You can define where to remove the name in *scope* using either the sheet index (counting begins at 0) or the following constants:

- `vk current sheet`
- `vk workbook`

## Example

```
$range:=VP Cell("ViewProArea";0;0)
VP ADD RANGE NAME("Total1";$range)

VP REMOVE NAME("ViewProArea";"Total1")
$formula:=VP Get formula by name("ViewProArea";"Total1")
//$formula=null
```

## See also

[VP Name](#)

## VP REMOVE SHEET

**VP REMOVE SHEET** ( *vpAreaName* : Text ; *index*: Integer )

Parameter	Type	Description
<i>vpAreaName</i>	Text	-> 4D View Pro area form object name
<i>index</i>	Integer	-> Index of the sheet to remove

## See also

[VP ADD SHEET](#)

## Description

The `VP REMOVE SHEET` command removes the sheet with the specified *index* from the document loaded in *vpAreaName*.

In *vpAreaName*, pass the name of the 4D View Pro area.

In *index*, pass the index of the sheet to remove. If the passed *index* does not exist, the command does nothing.

Indexing starts at 0.

## Example

The document currently has three sheets:



Remove the third sheet:

```
VP REMOVE SHEET("ViewProArea";2)
```



## VP REMOVE SPAN

**VP REMOVE SPAN** ( *rangeObj* : Object )

Parameter	Type	Description
<i>rangeObj</i>	Object->Range object	

### Description

The `VP REMOVE SPAN` command removes the span from the cells in *rangeObj*.

In *rangeObj*, pass a range object of the cell span. The spanned cells in the range are divided into individual cells.

### Example

To remove all cell spans from this document:

```
//find all cell spans
$span:=VP Get spans(VP All("ViewProArea"))

//remove the cell spans
VP REMOVE SPAN($span)
```

Result:

### See also

[VP ADD SPAN](#)  
[VP Get spans](#)

## VP REMOVE STYLESHEET

**VP REMOVE STYLESHEET** ( *vpAreaName* : Text ; *styleName* : Text { ; *sheet* : Integer } )

Parameter	Type	Description
<i>vpAreaName</i>	Text	->4D View Pro area form object name
<i>styleName</i>	Text	->Name of style to remove
<i>sheet</i>	Integer->	Sheet index (current sheet if omitted)

### Description

The `VP REMOVE STYLESHEET` command removes the style sheet passed in the *styleName* from the *vpAreaName*.

In `vpAreaName`, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

Pass the style sheet to remove in the `styleName` parameter.

You can define where to remove the style in the optional `sheet` parameter using the sheet index (counting begins at 0) or with the following constants:

- `vk current sheet`
- `vk workbook`

## Example

To remove the `GreenDashDotStyle` style object from the current sheet:

```
VP REMOVE STYLESHEET ("ViewProArea"; "GreenDashDotStyle")
```

## See also

[VP ADD STYLESHEET](#)

[VP Get stylesheet](#)

[VP Get stylesheets](#)

## VP REMOVE TABLE

- History

**VP REMOVE TABLE** ( `vpAreaName` : Object; `tableName` : Text {; `options` : Integer} {; `sheet` : Integer} )

Parameter	Type	Description
<code>vpAreaName</code>	Text	-> View Pro area name
<code>tableName</code>	Text	-> Name of the table to remove
<code>options</code>	Integer	-> Additional options
<code>sheet</code>	Integer	-> Sheet index (current sheet if omitted)

## Description

The `VP REMOVE TABLE` command removes a table that you created with [VP CREATE TABLE](#).

In `vpAreaName`, pass the name of the area where the table to remove is located.

In `tableName`, pass the name of the table to remove.

In `options`, you can specify additional behavior. Possible values are:

Constant	Value	Description
<code>vk table remove all</code>	0	Remove all including style and data
<code>vk table remove style1</code>		Remove style but keep data
<code>vk table remove data2</code>		Remove data but keep style

Table names are defined at sheet level. You can specify where the table is located using the optional `sheet` parameter (indexing starts at 0).

## Example

To remove the "people" table in the second sheet and keep the data in the cells:

```
VP REMOVE TABLE("ViewProArea"; "people"; vk table remove style; 2)
```

## See also

[VP CREATE TABLE](#)

## VP REMOVE TABLE COLUMNS

- History

**VP REMOVE TABLE COLUMNS** ( *vpAreaName* : Text ; *tableName* : Text ; *column* : Integer {; *count* : Integer {; *sheet* : Integer }}}

Parameter	Type	Description
vpAreaName	Text	->4D View Pro area form object name
tableName	Text	->Table name
column	Integer	->Index in the table of the starting column to remove
count	Text	->Number of columns to remove (must be >0)
sheet	Integer	->Sheet index (current sheet if omitted)

### Description

The `VP REMOVE TABLE COLUMNS` command removes one or *count* column(s) in the specified *tableName* at the specified *column* index. The command removes values and styles.

The command removes columns from the *tableName* table, NOT from the sheet. The total number of columns of the sheet is not impacted by the command. Data present at the right of the table (if any) are automatically moved left according to the number of removed columns.

If *tableName* does not exist, nothing happens.

### Example

To remove two columns from 3rd column of the "dataTable" table:

```
VP REMOVE TABLE COLUMNS("ViewProArea"; "dataTable"; 3; 2)
```

## See also

[VP INSERT TABLE COLUMNS](#)

[VP REMOVE TABLE ROWS](#)

## VP REMOVE TABLE ROWS

- History

**VP REMOVE TABLE ROWS** ( *vpAreaName* : Text ; *tableName* : Text ; *row* : Integer {; *count* : Integer {; *sheet* : Integer }})

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
vpAreaName	Text	-> 4D View Pro area form object name
tableName	Text	-> Table name
row	Integer	-> Index in the table of the starting row to remove
count	Text	-> Number of rows to remove (must be >0)
sheet	Integer	-> Sheet index (current sheet if omitted)

## Description

The `VP REMOVE TABLE ROWS` command removes one or *count* row(s) from the specified *tableName* at the specified *row* index. The command removes values and styles.

This command removes rows from the *tableName* table, NOT from the sheet. The total number of rows of the sheet is not impacted by the command. Data present below the table (if any) are automatically moved up according to the number of removed rows.

If the *tableName* table is bound to a [data context](#), the command removes element(s) from the collection.

If *tableName* does not exist, nothing happens.

## Example

To remove two rows from 3rd row of the "dataTable" table:

```
VP REMOVE TABLE ROWS ("ViewProArea"; "dataTable"; 3; 2)
```

## See also

[VP INSERT TABLE ROWS](#)

[VP REMOVE TABLE COLUMNS](#)

## VP RESET SELECTION

**VP RESET SELECTION** ( *vpAreaName* : Text { ; *sheet* : Integer } )

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
vpAreaName	Text	-> 4D View Pro area form object name
sheet	Integer	-> Sheet index (current sheet if omitted)

## Description

The `VP RESET SELECTION` command deselects all cells, resulting in no current selection or visible active cell.

A default active cell (cell A1) remains defined for 4D View Pro commands.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

In the optional *sheet* parameter, you can designate a specific spreadsheet where the range will be defined (counting begins at 0). If omitted, the current spreadsheet is used by default. You can explicitly select the current spreadsheet with the following constant:

- `vk current sheet`

## Example

You want to deselect all cells (the active cell and any selected cells):

```
VP RESET SELECTION ("myVPArea")
```

## See also

[VP ADD SELECTION](#)

[VP Get active cell](#)

[VP Get selection](#)

[VP SET ACTIVE CELL](#)

[VP SET SELECTION](#)

[VP SHOW CELL](#)

## VP RESIZE TABLE

- History

**VP RESIZE TABLE** ( *rangeObj* : Object; *tableName* : Text )

Parameter	Type	Description
<i>rangeObj</i>	Object ->	New range for the table
<i>tableName</i>	Text	-> Name of the table

## Description

The `VP RESIZE TABLE` command changes the *tableName* size with regards to the *rangeObj*.

The following rules apply:

- Headers must remain in the same row and the resulting table range must overlap the original table range.
- If the row count of the resized table is inferior to the initial row count, values inside cropped rows or columns are kept if they were not bound to a [data context](#), otherwise they are deleted.
- If the table expands on cells containing data:
  - if rows are added, data is deleted,
  - if columns are added, data are kept and are displayed in new columns.

If *tableName* does not exist, nothing happens.

## Example

You create a table with a data context:

```

var $context : Object
$context:=New object()

$context.col:=New collection
$context.col.push(New object("name"; "Smith"; "salary"; 10000))
$context.col.push(New object("name"; "Wesson"; "salary"; 50000))
$context.col.push(New object("name"; "Gross"; "salary"; 10500))

VP SET DATA CONTEXT("ViewProArea"; $context)

VP CREATE TABLE(VP Cells("ViewProArea"; 1; 1; 3; 3); "PeopleTable";
"col")

```

	A	B	C	D
1				
2	name	salary		
3	Smith	10000		
4	Wesson	50000		
5	Gross	10500		
6				
7				
8				

You want to add one column before and after the table as well as two empty rows.  
You can write:

```
VP RESIZE TABLE(VP Cells("ViewProArea"; 0; 1; 4; 6); "PeopleTable")
```

	A	B	C	D	E
1					
2	Column	name	salary	Column	
3		Smith	10000		
4		Wesson	50000		
5		Gross	10500		
6					
7					
8					

## See also

[VP CREATE TABLE](#)  
[VP Get table range](#)

## VP RESUME COMPUTING

**VP RESUME COMPUTING** ( *vpAreaName* : Text )

Parameter	Type	Description
<i>vpAreaName</i>	Text ->	4D View Pro area form object name

### Description

The `VP RESUME COMPUTING` command restarts the calculation of formulas in *vpAreaName*.

The command reactivates the calculation service in 4D View Pro. Any formulas impacted by changes made while calculations were suspended are updated, and formulas added after `VP RESUME COMPUTING` is executed are calculated.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

The 4D View Pro calculation service maintains a counter of suspend/resume actions. Therefore, each execution of `VP RESUME COMPUTING` must be balanced by a corresponding execution of the [VP SUSPEND COMPUTING](#) command.

## Example

See example in [VP SUSPEND COMPUTING](#).

## See also

[VP RECOMPUTE FORMULAS](#)

[VP SUSPEND COMPUTING](#)

## VP Row

**VP Row** ( *vpAreaName* : Text; *row* : Integer { ; *rowCount* : Integer { ; *sheet* : Integer } } ) : Object

Parameter	Type	Description
<i>vpAreaName</i>	Text	4D View Pro area form object name
<i>row</i>	Integer->	Row index
<i>rowCount</i>	Integer->	Number of rows

|*sheet* |Integer|->|Sheet index (current sheet if omitted)| |Result|Object|<-|Range object of row(s)|

## Description

The `VP Row` command returns a new range object referencing a specific row or rows.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

The *row* parameter defines the first row of the row range. Pass the row index (counting begins at 0) in this parameter. If the range contains multiple rows, you should also use the optional *rowCount* parameter.

The optional *rowCount* parameter allows you to define the total number of rows of the range. *rowCount* must be greater than 0. If omitted, the value will be set to 1 by default.

In the optional *sheet* parameter, you can designate a specific spreadsheet where the range will be defined (counting begins at 0). If not specified, the current spreadsheet is used by default. You can explicitly select the current spreadsheet with the following constant:

- `vk current sheet`

## Example

You want to define a range object for the row shown below (on the current spreadsheet):

You can write:

```
$row:=VP Row("ViewProArea";9) // row 10
```

## See also

[VP All](#)  
[VP Cell](#)  
[VP Cells](#)  
[VP Column](#)  
[VP Combine ranges](#)  
[VP Name](#)

## VP ROW AUTOFIT

**VP ROW AUTOFIT** (*rangeObj* : Object)

Parameter	Type	Description
-----------	------	-------------

rangeObj	Object->Range object	
----------	----------------------	--

### Description

The `VP ROW AUTOFIT` command automatically sizes the row(s) in *rangeObj* according to their contents.

In *rangeObj*, pass a range object containing a range of the rows whose size will be automatically handled.

### Example

The following rows don't correctly display the text:

```
VP ROW AUTOFIT(VP Row("ViewProArea";1;2))
```

Result:

## See also

[VP Column autofit](#)

## VP Run offscreen area

**VP Run offscreen area** (*parameters* : Object) : Mixed

Parameter	Type	Description
parameters	Object	Object containing the offscreen area's attributes
Result	Mixed	<- <code>.result</code> property of the <code>.onEvent</code> object, or Null if does not return a value

### Description

The `VP Run offscreen area` command creates an offscreen area in memory which can be used to process 4D View Pro area commands and functions.

In *parameters* object, pass any of the following optional properties. These properties will be available through the `This` command within the `onEvent` method and

reference the instance:

Property	Type	Description
area	text	The name of the offscreen area. If omitted or null, a generic name is assigned (e.g., "OffscreenArea1").
onEvent	object (formula)	A callback method that will be launched when the offscreen area is ready. It can be either: <ul style="list-style-type: none"><li>• an <code>onEvent</code> function of a class, or</li><li>• a <code>Formula</code> object</li></ul> By default, the callback method is called on the <code>On VP Ready</code> , <code>On Load</code> , <code>On Unload</code> , <code>On End URL Loading</code> , <code>On URL Loading Error</code> , <code>On VP Range Changed</code> , or <code>On Timer</code> events. The callback method can be used to access the <a href="#">4D View Pro form object variable</a> .
autoQuit	boolean	True (default value) if the command must stop the formula execution when the <code>On End URL Loading</code> or <code>On URL Loading Error</code> events occur. If false, you must use the <code>CANCEL</code> or <code>ACCEPT</code> commands in the <code>onEvent</code> callback method.
timeout	number	Maximum time (expressed in seconds) before the area automatically closes if no event is generated. If set to 0, no limitation is applied. Default value: 60
result	mixed	Result of the processing (if any)
<code>&lt;customProperty&gt;</code>	mixed	Any custom attribute to be available in the <code>onEvent</code> callback method.

The following property is automatically added by the command if necessary:

Property	Type	Description
timeoutReached	boolean	Added with true value if timeout has been exceeded

The offscreen area is only available during the execution of the `VP Run offscreen area` command. It will automatically be destroyed once execution has ended.

The following commands can be used in the callback method:

- `ACCEPT`
- `CANCEL`
- `SET TIMER`
- `WA Evaluate JavaScript`
- `WA EXECUTE JAVASCRIPT FUNCTION`

## Example 1

You want to create an offscreen 4D View Pro area and get the value of a cell:

```

// cs.OffscreenArea class declaration
Class constructor ($path : Text)
  This.filePath:=$path

// This function will be called on each event of the offscreen area
Function onEvent()
  Case of
    :(FORM Event.code=On VP Ready)
      VP IMPORT DOCUMENT(This.area;This.filePath)
      This.result:=VP Get value(VP Cell(This.area;6;22))

      ALERT("The G23 cell contains the value: "+String(This.result))
    End case

```

### The *OffscreenArea* callback method:

```

$o:=cs.OffscreenArea.new()
$result:=VP Run offscreen area($o)

```

### Example 2

You want to load a large document offscreen, wait for all calculations to complete evaluating, and export it as a PDF:

```

//cs.OffscreenArea class declaration
Class constructor($pdfPath : Text)
  This.pdfPath:=$pdfPath
  This.autoQuit:=False
  This.isWaiting:=False

Function onEvent()
  Case of
    :(FORM Event.code=On VP Ready)
    // Document import
    VP IMPORT DOCUMENT(This.area;$largeDocument4VP)
    This.isWaiting:=True

    // Start a timer to verify if all calculations are finished.
    // If during this period the "On VP Range Changed" is thrown, the
    timer will be restarted
    // The time must be defined according to the computer configuration.
    SET TIMER(60)

    :(FORM Event.code=On VP Range Changed)
    // End of calculation detected. Restarts the timer
    If(This.isWaiting)
      SET TIMER(60)
    End if

    :(FORM Event.code=On Timer)
    // To be sure to not restart the timer if you call others 4D View
    command after this point
    This.isWaiting:=False

    // Stop the timer
    SET TIMER(0)

    // Start the PDF export
    VP EXPORT DOCUMENT(This.area;This.pdfPath;New
object("formula";Formula(ACCEPT)))

    :(FORM Event.code=On URL Loading Error)
      CANCEL
End case

```

The `OffscreenArea` callback method:

```
$o:=cs.OffscreenArea.new()  
  
$result:=VP Run offscreen area($o)
```

## See also

[Blog post: End of document loading](#)

# S

## VP SET ACTIVE CELL

**VP SET ACTIVE CELL** (*rangeObj* : Object)

Parameter	Type	Description
-----------	------	-------------

rangeObj	Object->Range object	
----------	----------------------	--

### Description

The `VP SET ACTIVE CELL` command defines a specified cell as active.

In *rangeObj*, pass a range containing a single cell as an object (see [VP Cell](#)). If *rangeObj* is not a cell range or contains multiple ranges, the first cell of the first range is used.

### Example

To set the cell in column D, row 5 as the active cell:

```
$activeCell:=VP Cell("myVPArea";3;4)  
VP SET ACTIVE CELL($activeCell)
```

## See also

[VP ADD SELECTION](#)

[VP Get active cell](#)

[VP Get selection](#)

[VP RESET SELECTION](#)

[VP SET SELECTION](#)

[VP SHOW CELL](#)

## VP SET ALLOWED METHODS

**VP SET ALLOWED METHODS** (*methodObj* : Object)

Parameter	Type	Description
-----------	------	-------------

methodObj	Object->	Allowed methods in the 4D View Pro areas
-----------	----------	--

### Compatibility

For greater flexibility, it is recommended to use the [VP SET CUSTOM FUNCTIONS](#) command which allows you to designate 4D formulas that can be called from 4D View Pro areas. As soon as `VP SET CUSTOM FUNCTIONS` is called, `VP SET ALLOWED METHODS` calls are ignored. 4D View Pro also

supports 4D's generic `SET ALLOWED METHODS` command if neither `VP SET CUSTOM FUNCTIONS` nor `VP SET ALLOWED METHODS` are called, however using the generic command is not recommended.

## Description

The `VP SET ALLOWED METHODS` command designates the project methods that can be called in 4D View Pro formulas. This command applies to all 4D View Pro areas initialized after its call during the session. It can be called multiple times in the same session to initialize different configurations.

By default for security reasons, if you do not execute the `VP SET ALLOWED METHODS` command, no method call is allowed in 4D View Pro areas -- except if 4D's generic `SET ALLOWED METHODS` command was used (see compatibility note). Using an unauthorized method in a formula prints a #NAME? error in the 4D View Pro area.

In the *methodObj* parameter, pass an object in which each property is the name of a function to define in the 4D View Pro areas:

Property	Type	Description
<functionName>	Object	Custom function definition. The <functionName> property name defines the name of the custom function to display in 4D View Pro formulas (no spaces allowed)
method	Text	(mandatory) Name of the existing 4D project method to allow
parameters	Collection of objects	Collection of parameters (in the order they are defined in the method).
[ ].name	Text	Name of a parameter to display for the <functionName>. <b>Note:</b> Parameter names must not contain space characters.
[ ].type	Number	Type of the parameter. Supported types: <ul style="list-style-type: none"> <li>• Is Boolean</li> <li>• Is date</li> <li>• Is Integer</li> <li>• Is object</li> <li>• Is real</li> <li>• Is text</li> <li>• Is time</li> </ul> If omitted, by default the value is automatically sent with its type, except date or time values which are sent as an object (see <a href="#">Parameters</a> section). If type is Is object, the object has the same structure as the object returned by <a href="#">VP Get value</a> .
summary	Text	Function description to display in 4D View Pro
minParams	Number	Minimum number of parameters
maxParams	Number	Maximum number of parameters. Passing a number higher than the length of parameters allows declaring "optional" parameters with default type

## Example

You want to allow two methods in your 4D View Pro areas:

```

C_OBJECT($allowed)
$allowed:=New object //parameter for the command

$allowed.Hello:=New object //create a first simple function named
"Hello"
$allowed.Hello.method:="My_Hello_Method" //sets the 4D method
$allowed.Hello.summary:="Hello prints hello world"

$allowed.Byebye:=New object //create a second function with parameters
named "Byebye"
$allowed.Byebye.method:="My_ByeBye_Method"
$allowed.Byebye.parameters:=New collection
$allowed.Byebye.parameters.push(New object("name"; "Message"; "type"; Is
text))
$allowed.Byebye.parameters.push(New object("name"; "Date"; "type"; Is
date))
$allowed.Byebye.parameters.push(New object("name"; "Time"; "type"; Is
time))
$allowed.Byebye.summary:="Byebye prints a custom timestamp"
$allowed.Byebye.minParams:=3
$allowed.Byebye.maxParams:=3

```

VP SET ALLOWED METHODS (\$allowed)

After this code is executed, the defined functions can be used in 4D View Pro formulas:

	A	B	C	D	E
1	=BYEBYE(				
2	BYEBYE(Message; Date; Time)				
3	Summary				
4	Byebye prints a custom timestamp				
5					

In 4D View Pro formulas, function names are automatically displayed in uppercase.

## See also

[4D functions](#)

[VP SET CUSTOM FUNCTIONS](#)

## VP SET BINDING PATH

- History

**VP SET BINDING PATH** ( *rangeObj* : Object ; *dataContextAttribute* : Text)

Parameter	Type	Description
<i>rangeObj</i>	Object-> Range object	
<i>dataContextAttributeText</i>	-> Name of the attribute to bind to <i>rangeObj</i>	

## Description

The `VP SET BINDING PATH` command binds an attribute from a sheet's data context to *rangeObj*. After you set a data context using the [SET DATA CONTEXT](#) method. When loaded, if the data context contains the attribute, the value of *dataContextAttribute* is automatically displayed in the cells in *rangeObj*.

In *rangeObj*, pass an object that is either a cell range or a combined range of cells.

- If *rangeObj* is a range with several cells, the command binds the attribute to the first cell of the range.
- If *rangeObj* contains several ranges of cells, the command binds the attribute to the first cell of each range.

In *dataContextAttribute*, pass the name of the attribute to bind to *rangeObj*. If *dataContextAttribute* is an empty string, the function removes the current binding.

Attributes of type collection are not supported. When you pass the name of a collection attribute, the command does nothing.

## Example

Set a data context and bind the `firstName` and `lastName` attribute to cells:

```
var $p : Object
$p:=New object
$p.firstName:="Freehafer"
$p.lastName:="Nancy"

VP SET DATA CONTEXT ("ViewProArea"; $p)

VP SET BINDING PATH(VP Cell("ViewProArea"; 0; 0); "firstName")
VP SET BINDING PATH(VP Cell("ViewProArea"; 1; 0); "lastName")
```

	0	1	2	3
1	Freehafer	Nancy		
2				
3				

## See also

[VP Get binding path](#)  
[VP Get data context](#)  
[VP SET DATA CONTEXT](#)

## VP SET BOOLEAN VALUE

**VP SET BOOLEAN VALUE** (*rangeObj* : Object ; *boolValue* : Boolean)

Parameter	Type	Description
<i>rangeObj</i>	Object	-> Range object
<i>boolValue</i>	Boolean	-> Boolean value to set

## Description

The `VP SET BOOLEAN VALUE` command assigns a specified boolean value to a designated cell range.

In *rangeObj*, pass a range of the cell(s) (created for example with [VP Cell](#) or [VP Column](#)) whose value you want to specify. If *rangeObj* includes multiple cells, the value specified will be repeated in each cell.

The *boolValue* parameter allows you to pass the boolean value (**True** or **False**) that will be assigned to the *rangeObj*.

## Example

```
//Set the cell value as False  
VP SET BOOLEAN VALUE(VP Cell("ViewProArea";3;2);False)
```

## See also

[VP SET VALUE](#)

## VP SET BORDER

**VP SET BORDER** (*rangeObj* : Object ; *borderStyleObj* : Object ; *borderPosObj* : Object )

Parameter	Type	Description
<i>rangeObj</i>	Object -> Range object	
<i>borderStyleObj</i>	Object -> Object containing border line style	
<i>borderPosObj</i>	Object -> Object containing border placement	

## Description

The `VP SET BORDER` command applies the border style(s) defined in *borderStyleObj* and *borderPosObj* to the range defined in the *rangeObj*.

In *rangeObj*, pass a range of cells where the border style will be applied. If the *rangeObj* contains multiple cells, borders applied with `VP SET BORDER` will be applied to the *rangeObj* as a whole (as opposed to the [VP SET CELL STYLE](#) command which applies borders to each cell of the *rangeObj*). If a style sheet has already been applied, `VP SET BORDER` will override the previously applied border settings for the *rangeObj*.

The *borderStyleObj* parameter allows you to define the style for the lines of the border. The *borderStyleObj* supports the following properties:

Property	Type	Description	Possible values
<i>color</i>	text	Defines the color of the border. Default = black.	CSS color "#rrggbb" syntax (preferred syntax), CSS color "rgb(r,g,b)" syntax (alternate syntax), CSS color name (alternate syntax) <ul style="list-style-type: none"><li>• <code>vk line style dash dot</code></li><li>• <code>vk line style dash dot dot</code></li><li>• <code>vk line style dashed</code></li><li>• <code>vk line style dotted</code></li><li>• <code>vk line style double</code></li><li>• <code>vk line style empty</code></li><li>• <code>vk line style hair</code></li><li>• <code>vk line style medium</code></li><li>• <code>vk line style medium dash dot</code></li><li>• <code>vk line style medium dash dot dot</code></li><li>• <code>vk line style medium dashed</code></li><li>• <code>vk line style slanted dash dot</code></li><li>• <code>vk line style thick</code></li><li>• <code>vk line style thin</code></li></ul>
<i>style</i>	Integer	Defines the style of the border. Default = empty.	

You can define the position of the *borderStyleObj* (i.e., where the line is applied) with the *borderPosObj*:

Property	Type	Description
all	boolean	Border line style applied to all borders.
left	boolean	Border line style applied to left border.
top	boolean	Border line style applied to top border.
right	boolean	Border line style applied to right border.
bottom	boolean	Border line style applied to bottom border.
outline	boolean	Border line style applied to outer borders only.
inside	boolean	Border line style applied to inner borders only.
innerHorizontal	boolean	Border line style applied to inner horizontal borders only.
innerVertical	boolean	Border line style applied to inner vertical borders only.

## Example 1

This code produces a border around the entire range:

```
$border:=New object("color";"red";"style";vk line style thick)
$option:=New object("outline";True)
VP SET BORDER(VP Cells("ViewProArea";1;1;3;3);$border;$option)
```

	A	B	C	D	E
1					
2					
3					
4					
5					
6					

## Example 2

This code demonstrates the difference between `VP SET BORDER` and setting borders with the [VP SET CELL STYLE](#) command:

```
// Set borders using VP SET BORDER
$border:=New object("color";"red";"style";vk line style thick)
$option:=New object("outline";True)
VP SET BORDER(VP Cells("ViewProArea";1;1;3;3);$border;$option)

// // Set borders using VP SET CELL STYLE
$cCellStyle:=New object
$cCellStyle.borderBottom:=New object("color";"blue";"style";vk line
style thick)
$cCellStyle.borderRight:=New object("color";"blue";"style";vk line style
thick)
VP SET CELL STYLE(VP Cells("ViewProArea";4;4;3;3);$cCellStyle)
```

	A	B	C	D	E	F	G	H
1								
2								
3								
4								
5								
6								
7								
8								

## See also

[VP SET CELL STYLE](#)

## VP SET CELL STYLE

**VP SET CELL STYLE** (*rangeObj* : Object ; *styleObj* : Object)

### Parameter Type Description

*rangeObj* Object->Range object  
*styleObj* Object->Style object

### Description

The `VP SET CELL STYLE` command applies the style(s) defined in the *styleObj* to the cells defined in the *rangeObj*.

In *rangeObj*, pass a range of cells where the style will be applied. If the *rangeObj* contains multiple cells, the style is applied to each cell.

Borders applied with `VP SET CELL STYLE` will be applied to each cell of the *rangeObj*, as opposed to the [VP SET BORDER](#) command which applies borders to the *rangeObj* as a whole.

The *styleObj* parameter lets you pass an object containing style settings. You can use an existing style sheet or create a new style. If the *styleObj* contains both an existing style sheet and additional style settings, the existing style sheet is applied first, followed by the additional settings.

To remove a style and revert to the default style settings (if any), pass a NULL value:

- giving the *styleObj* parameter a NULL value will remove any style settings from the *rangeObj*,
- giving an attribute a NULL value will remove this specific attribute from the *rangeObj*.

For more information about style objects and style sheets, see the [Style Objects](#) paragraph.

### Example

```
$style:=New object
$style.font:="8pt Arial"
$style.backColor:="Azure"
$style.foreColor:="red"
$style.hAlign:=1
$style.isVerticalText:=True
$style.borderBottom:=New object("color";"#800080";"style";vk line style
thick)
$style.backgroundImage:=Null //remove a specific attribute

VP SET CELL STYLE(VP Cell("ViewProArea";1;1);$style)
```

	A	B	C
1			
2		H e l l o  W o r l d	
3			

## See also

[VP ADD STYLESHEET](#)

[VP Font to object](#)

[VP Get cell style](#)

[VP Object to font](#)

[VP SET BORDER](#)

[VP SET DEFAULT STYLE](#)

## VP SET COLUMN ATTRIBUTES

**VP SET COLUMN ATTRIBUTES** (*rangeObj* : Object ; *propertyObj* : Object)

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
------------------	-------------	--------------------

*rangeObj* Object->Range object

*propertyObj* Object->Object containing column properties

### Description

The `VP SET COLUMN ATTRIBUTES` command applies the attributes defined in the *propertyObj* to the columns in the *rangeObj*.

In *rangeObj*, pass an object containing a range. If the range contains both columns and rows, attributes are applied only to the columns.

The *propertyObj* parameter lets you specify the attributes to apply to the columns in the *rangeObj*. These attributes are:

<b>Property</b>	<b>Type</b>	<b>Description</b>
width	number	Column width expressed in pixels
pageBreak	boolean	True to insert a page break before the first column of the range, else false
visible	boolean	True if the column is visible, else false
resizable	boolean	True if the column can be resized, else false
header	text	Column header text

### Example

To change the size of the second column and set the header, you write:

```
C_OBJECT($column;$properties)
```

```
$column:=VP Column("ViewProArea";1) //column B
$properties:=New object("width";100;"header";"Hello World")
```

```
VP SET COLUMN ATTRIBUTES($column;$properties)
```

	A	Hello World	C	D	E	F	G	H	I
1	The	quick	brown	fox	jumped	over	the	lazy	dog

## See also

[VP Column](#)

[VP Get column attributes](#)

[VP Get row attributes](#)

[VP SET ROW ATTRIBUTES](#)

## VP SET COLUMN COUNT

**VP SET COLUMN COUNT** ( *vpAreaName* : Text , *columnCount* : Integer { , *sheet* : Integer } )

Parameter	Type	Description
<i>vpAreaName</i>	Text	->4D View Pro area form object name
<i>columnCount</i>	Integer->	Number of columns
<i>sheet</i>	Integer->	Sheet index (current sheet if omitted)

## Description

The `VP SET COLUMN COUNT` command defines the total number of columns in *vpAreaName*.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

Pass the total number of columns in the *columnCount* parameter. *columnCount* must be greater than 0.

In the optional *sheet* parameter, you can designate a specific spreadsheet where the *columnCount* will be applied (counting begins at 0). If omitted, the current spreadsheet is used by default. You can explicitly select the current spreadsheet with the following constant:

- `vk current sheet`

## Example

The following code defines five columns in the 4D View Pro area:

```
VP SET COLUMN COUNT ("ViewProArea";5)
```

## See also

[VP Get column count](#)

[VP Get row count](#)

[VP SET ROW COUNT](#)

## VP SET CURRENT SHEET

**VP SET CURRENT SHEET** ( *vpAreaName* : Text ; *sheet* : Integer )

Parameter	Type	Description
vpAreaNameText	-> 4D View Pro area form object name	
sheet	Integer <- Index of the new current sheet	

## Description

The `VP SET CURRENT SHEET` command sets the current sheet in `vpAreaName`. The current sheet is the selected sheet in the document.

In `vpAreaName`, pass the name of the 4D View Pro area.

In `sheet`, pass the index of the sheet to be set as current sheet. If the index passed is inferior to 0 or exceeds the number of sheets, the command does nothing.

Indexing starts at 0.

## Example

The document's current sheet is the first sheet:



Set the current sheet to the third sheet:

```
VP SET CURRENT SHEET("ViewProArea";2)
```



## See also

[VP Get current sheet](#)

## VP SET CUSTOM FUNCTIONS

**VP SET CUSTOM FUNCTIONS** ( `vpAreaName` : Text ; `formulaObj` : Object )

Parameter	Type	Description
vpAreaNameText	-> 4D View Pro area form object name	
formulaObj	Object -> Formula object	

## Description

The `VP SET CUSTOM FUNCTIONS` command designates the 4D formulas that can be called directly from 4D View Pro formulas. Because custom functions are not stored in the document, `VP SET CUSTOM FUNCTIONS` must be executed in the `On Load` form event.

The formulas specified by `VP SET CUSTOM FUNCTIONS` appear in a pop-up menu when the first letter of their name is entered. See the [Formulas and Functions](#) page.

If `VP SET CUSTOM FUNCTIONS` is called multiple times for the same area, in the same session, only the last call is taken into account.

Pass the name of the 4D View Pro area in `vpAreaName`. If you pass a name that does

not exist, an error is returned.

In the `formulaObj` parameter, pass an object containing the 4D formulas that can be called from 4D View Pro formulas as well as additional properties. Each `customFunction` property passed in `formulaObj` becomes the name of a function in the 4D View Pro area.

Property	Type	Description
<code>&lt;customFunction&gt;</code>	Object	Custom function definition. <code>&lt;customFunction&gt;</code> defines the name of the custom function to display in 4D View Pro formulas (no spaces allowed)
<code>formula</code>	Object	4D formula object (mandatory). See the <code>Formula</code> command.
<code>parameters</code>	Collection of objects	Collection of parameters (in the order they are defined in the formula)
<code>[ ].name</code>	Text	Name of parameter to display in 4D View Pro
<code>[ ].type</code>	Number	Type of the parameter. Supported types: <ul style="list-style-type: none"><li>• Is Boolean</li><li>• Is date</li><li>• Is Integer</li><li>• Is object</li><li>• Is real</li><li>• Is text</li><li>• Is time</li></ul> If <code>type</code> is omitted or if the default value (-1) is passed, the value is automatically sent with its type, except date or time values which are sent as an object (see <a href="#">Parameters</a> section). If <code>type</code> is <code>Is object</code> , the object has the same structure as the object returned by <a href="#">VP Get value</a> .
<code>summary</code>	Text	Formula description to display in 4D View Pro
<code>minParams</code>	Number	Minimum number of parameters
<code>maxParams</code>	Number	Maximum number of parameters. Passing a number higher than the length of <code>parameters</code> allows declaring "optional" parameters with default type

## WARNING

- As soon as `VP SET CUSTOM FUNCTIONS` is called, the methods allowed by the [VP SET ALLOWED METHODS](#) command (if any) are ignored in the 4D View Pro area.
- As soon as `VP SET CUSTOM FUNCTIONS` is called, the functions based upon `SET TABLE TITLES` and `SET FIELD TITLES` commands are ignored in the 4D View Pro area.

## Example

You want to use formula objects in a 4D View Pro area to add numbers, retrieve a customer's last name and gender:

```
Case of
  : (FORM Event.code=On Load)

    var $o : Object
    $o:=New object

    // Define "addnum" function from a method named "addnum"
    $o.addnum:=New object
    $o.addnum.formula:=Formula(addnum)
    $o.addnum.parameters:=New collection
    $o.addnum.parameters.push(New object("name";"num1";"type";Is
Integer))
    $o.addnum.parameters.push(New object("name";"num2";"type";Is
Integer))

    // Define "ClientLastName" function from a database field
    $o.ClientLastName:=New object
    $o.ClientLastName.formula:=Formula([Customers]lastname)
    $o.ClientLastName.summary:="Lastname of the current client"

    // Define "label" function from a 4D expression with one parameter
    $o.label:=New object
    $o.label.formula:=Formula(ds.Customers.get($1).label)
    $o.label.parameters:=New collection
    $o.label.parameters.push(New object("name";"ID";"type";Is
Integer))

    // Define "Title" function from a variable named "Title"
    $o.Title:=New object
    $o.Title.formula:=Formula(Title)

    VP SET CUSTOM FUNCTIONS("ViewProArea";$o)
```

End case

## See also

[VP SET ALLOWED METHODS](#)

## VP SET DATA CONTEXT

- History

**VP SET DATA CONTEXT** ( *vpAreaName* : Text ; *dataObj* : Object {; *options* : Object } {; *sheet* : Integer} )

**VP SET DATA CONTEXT** ( *vpAreaName* : Text ; *dataColl* : Collection ; {*options* : Object } {; *sheet* : Integer} )

Parameter	Type	Description
<i>vpAreaName</i>	Object	->4D View Pro area form object name
<i>dataObj</i>	Object	->Data object to load in the data context
<i>dataColl</i>	Collection	-> Data collection to load in the data context
<i>options</i>	Object	->Additional options
<i>sheet</i>	Integer	->Sheet index

## Description

The `VP SET DATA CONTEXT` command sets the data context of a sheet. A data context is an object or a collection bound to a worksheet, and whose contents can be used to automatically fill the sheet cells, either by using an autogenerate option or the [VP SET BINDING PATH](#) method. On the other hand, the [VP Get data context](#) command can return a context containing user modifications.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

In *dataObj* or *dataColl*, pass an object or a collection containing the data to load in the data context. Images are converted to data URI schemes.

To pass a time value in *dataObj* or *dataColl*, encapsulate it in an object with the following properties (see [example 4](#)):

Property	Type	Description
value	Integer, Real, Boolean, Text, Date, Null	Value to put in the context
time	Real	Time value (in seconds) to put in the context

In *options*, you can pass an object that specifies additional options. Possible properties are:

Property	Type	Description
reset	Object	True to reset the sheet's contents before loading the new context, False (default) otherwise. Only used when data is a collection. True (default) to specify that columns must be generated automatically when the data context is bound. In this case, the following rules apply:
autoGenerateColumnsObject		<ul style="list-style-type: none"><li>If <i>dataColl</i> is a collection of objects, attribute names are used as column titles (see <a href="#">example 2</a>).</li><li>If <i>dataColl</i> contains subcollections of scalar values, each subcollection defines the values in a row (see <a href="#">example 3</a>). The first subcollection determines how many columns are created.</li></ul>

In *sheet*, pass the index of the sheet that will receive the data context. If no index is passed, the context is applied to the current sheet.

If you export your document to an object using [VP Export to object](#), or to a 4DVP document using [VP EXPORT DOCUMENT](#), the `includeBindingSource` option lets you copy the contents of the current contexts as cell values in the exported object or document. For more details, refer to the description of those methods.

## Example

Pass an object and bind the context data to cells in the first row:

```

var $data : Object
$data:=New object
$data.firstName:="Freehafer"
$data.lastName:="Nancy"

VP SET DATA CONTEXT("ViewProArea"; $data)
VP SET BINDING PATH(VP Cell("ViewProArea"; 0; 0); "firstName")
VP SET BINDING PATH(VP Cell("ViewProArea"; 1; 0); "lastName")

```

	0	1	2	3
1	Freehafer	Nancy		
2				
3				

## Example 2

Pass a collection of objects and generate columns automatically:

```

var $options : Object
var $data : Collection

$data:=New collection()
$data.push(New object("firstname"; "John"; "lastname"; "Smith"))
$data.push(New object("firstname"; "Mary"; "lastname"; "Poppins"))

$options:=New object("autoGenerateColumns"; True)

VP SET DATA CONTEXT("ViewProArea"; $data; $options)

```

A1	:	X ✓ fx	
firstname	lastname		
1 John	Smith		
2 Mary	Poppins		

## Example 3

The *data* passed as a parameter is a collection that contains subcollections. Each subcollection defines the contents of a row:

```

var $data : Collection
var $options : Object

$data:=New collection
$data.push(New collection(1; 2; 3; False; "")) // 5 columns are
created
$data.push(New collection) // Second row is empty
$data.push(New collection(4; 5; Null; "hello"; "world")) // Third row
has 5 values
$data.push(New collection(6; 7; 8; 9)) // Fourth row has 4 values

$options:=New object("autoGenerateColumns"; True)

VP SET DATA CONTEXT("ViewProArea"; $data; $options)

```

0	1	2	3	4
1	1	2	3	FALSE
2				
3	4	5	hello	world
4	6	7	8	9

## Example 4 - Date and time syntax

```

var $data : Collection
var $options : Object

$data:= New collection()

// Dates can be passed as scalar values
$data.push(New collection("Date"; Current date))

// Time values must be passed as object attributes
$data.push(New collection("Time"; New object("time"; 5140)))

// Date + time example
$data.push(New collection("Date + Time"; New object("value"; Current
date; "time"; 5140)))

$options:=New object("autoGenerateColumns"; True)

VP SET DATA CONTEXT("ViewProArea"; $data; $options)

```

Here's the result once the columns are generated:

0	1
1	Date
2	Time
3	Date + Time

2/22/2022  
1:25:40 AM  
2/22/2022 01:25:40

## See also

[VP SET BINDING PATH](#)  
[VP Get binding path](#)  
[VP Get data context](#)

## VP SET DATE TIME VALUE

**VP SET DATE TIME VALUE** (*rangeObj* : Object ; *dateValue* : Date ; *timeValue* : Time ; *formatPattern* : Text )

Parameter	Type	Description
rangeObj	Object-> Range object	
dateValue	Date -> Date value to set	
timeValue	Time -> Time value to set	
formatPattern	Text -> Format of value	

## Description

The `VP SET DATE TIME VALUE` command assigns a specified date and time value to a designated cell range.

In *rangeObj*, pass a range of the cell(s) (created for example with [VP Cell](#) or [VP Column](#)) whose value you want to specify. If *rangeObj* includes multiple cells, the

value specified will be repeated in each cell.

The *dateValue* parameter specifies a date value to be assigned to the *rangeObj*.

The *timeValue* parameter specifies a time value (expressed in seconds) to be assigned to the *rangeObj*.

The optional *formatPattern* defines a pattern for the *dateValue* and *timeValue* parameters. For information on patterns and formatting characters, please refer to the [Date and time formats](#) section.

## Example

```
//Set the cell value as the current date and time
VP SET DATE TIME VALUE(VP Cell("ViewProArea";6;2);Current time;Current
date;vk pattern full date time)

//Set the cell value as the 18th of December
VP SET DATE TIME VALUE(VP Cell("ViewProArea";3;9);!2024-12-18!;?
14:30:10?;vk pattern sortable date time)
```

## See also

[4D View Pro cell format](#)  
[VP SET DATE VALUE](#)  
[VP SET TIME VALUE](#)  
[VP SET VALUE](#)

## VP SET DATE VALUE

**VP SET DATE VALUE** ( *rangeObj* : Object ; *dateValue* : Date { ; *formatPattern* : Text } )

|Parameter|Type||Description|

|---|---|---|---| |*rangeObj* |Object|->|Range object| |*dateValue* |Date|->|Date value
to set| |*formatPattern* |Text|->|Format of value|

## Description

The `VP SET DATE VALUE` command assigns a specified date value to a designated cell range.

In *rangeObj*, pass a range of the cell(s) whose value you want to specify. If *rangeObj* includes multiple cells, the value specified will be repeated in each cell.

The *dateValue* parameter specifies a date value to be assigned to the *rangeObj*.

The optional *formatPattern* defines a pattern for the *dateValue* parameter. Pass any custom format or you can use one of the following constants:

Constant	Description	Default US pattern
<code>vk pattern long date</code>	ISO 8601 format for the full date	"dddd, dd MMMM yyyy"
<code>vk pattern month day</code>	ISO 8601 format for the month and day	"MMMM dd"
<code>vk pattern short date</code>	Abbreviated ISO 8601 format for the date	"MM/dd/yyyy"
<code>vk pattern year month</code>	ISO 8601 format for the month and year	"yyyy MMMM"

For information on patterns and formatting characters, please refer to the [Date and time formats](#) section.

## Example

```
//Set the cell value to the current date
VP SET DATE VALUE(VP Cell("ViewProArea";4;2);Current date)

//Set the cell value to a specific date with a designated format
VP SET DATE VALUE(VP Cell("ViewProArea";4;4);Date("12/25/94");"d/m/yy")
)
VP SET DATE VALUE(VP Cell("ViewProArea";4;6);!2005-01-15!;vk pattern
month day)
```

## See also

[4D View Pro cell format](#)  
[VP SET DATE TIME VALUE](#)  
[VP SET VALUE](#)

## VP SET DEFAULT STYLE

**VP SET DEFAULT STYLE** ( *vpAreaName* : Text ; *styleObj* : Object { ; *sheet* : Integer } )

Parameter	Type	Description
<i>vpAreaName</i>	Text	->4D View Pro area form object name
<i>styleObj</i>	Object	->Style object
<i>sheet</i>	Integer	-> Sheet index (default = current sheet)

## Description

The `VP SET DEFAULT STYLE` command defines the style in the *styleObj* as the default style for a *sheet*.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

The *styleObj* lets you pass an object containing style settings. You can use an existing style sheet or you can create a new style. For more information, see the [Style objects](#) paragraph.

In the optional *sheet* parameter, you can designate a specific spreadsheet where the style will be defined. If omitted, the current spreadsheet is used by default. You can explicitly select the current spreadsheet with the following constant:

- `vk current sheet`

## Example

```
$style:=New object  
$style.hAlign:=vk horizontal align left  
$style.font:="12pt papyrus"  
$style.backColor:="#E6E6FA" //light purple color  
  
VP SET DEFAULT STYLE("myDoc";$style)
```

	A	B	C
1			
2		Hello World!	
3			
4			
5			
6			
7			
8			

## See also

[VP ADD STYLESHEET](#)

[VP Font to object](#)

[VP Get default style](#)

[VP Object to font](#)

[VP SET BORDER](#)

[VP SET CELL STYLE](#)

## VP SET FIELD

**VP SET FIELD** ( *rangeObj* : Object ; *field* : Pointer { ; *formatPattern* : Text } )

Parameter	Type	Description
<i>rangeObj</i>	Object -> Range object	
<i>field</i>	Pointer-> Reference to field in virtual structure	
<i>formatPattern</i>	Text -> Format of field	

## Description

The `VP SET FIELD` command assigns a 4D database virtual field to a designated cell range.

In *rangeObj*, pass a range of the cell(s) whose value you want to specify. If *rangeObj* includes multiple cells, the specified field will be linked in each cell.

The *field* parameter specifies a 4D database [virtual field](#) to be assigned to the *rangeObj*. The virtual structure name for *field* can be viewed in the formula bar. If any of the cells in *rangeObj* have existing content, it will be replaced by *field*.

The optional *formatPattern* defines a pattern for the *field* parameter. You can pass any valid [custom format](#).

## Example

```
VP SET FIELD(VP Cell("ViewProArea";5;2);->[TableName]Field)
```

## See also

[VP SET VALUE](#)

## VP SET FORMULA

**VP SET FORMULA** ( *rangeObj* : Object ; *formula* : Text { ; *formatPattern* : Text } )

### Parameter Type Description

|*rangeObj* |Object|->|Range object| |*formula* |Text|->|Formula or 4D method|  
|*formatPattern* |Text|->|Format of field|

### Description

The `VP SET FORMULA` command assigns a specified formula or 4D method to a designated cell range.

In *rangeObj*, pass a range of the cell(s) (created for example with [VP Cell](#) or [VP Column](#)) whose value you want to specify. If *rangeObj* includes multiple cells, the formula specified will be linked in each cell.

The *formula* parameter specifies a formula or 4D method name to be assigned to the *rangeObj*.

If the *formula* is a string, use the period `.` as numerical separator and the comma `,` as parameter separator. If a 4D method is used, it must be allowed with the [VP SET ALLOWED METHODS](#) command.

The optional *formatPattern* defines a [pattern](#) for the *formula*.

You remove the formula in *rangeObj* by replacing it with an empty string ("").

### Example 1

```
VP SET FORMULA(VP Cell("ViewProArea";5;2);"SUM($A$1:$C$10)")
```

### Example 2

To remove the formula:

```
VP SET FORMULA(VP Cell("ViewProArea";5;2); "")
```

### Example 3

```
VP SET FORMULA($range;"SUM(A1,B7,C11)") //", " to separate parameters
```

## See also

[Cell format](#)

[VP Get Formula](#)

[VP SET FORMULAS](#)

[VP SET VALUE](#)

## VP SET FORMULAS

**VP SET FORMULAS** ( *rangeObj* : Object ; *formulasCol* : Collection )

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
rangeObj	Object	-> Cell range object
formulasCol	Collection	-> Collection of formulas

## Description

The `VP SET FORMULAS` command assigns a collection of formulas starting at the specified cell range.

In *rangeObj*, pass a range of the cell (created with [VP Cell](#)) whose formula you want to specify. If *rangeObj* includes multiple ranges, only the first range is used.

The *formulasCol* is a two-dimensional collection:

- The first-level collection contains subcollections of formulas. Each subcollection defines a row.
- Each subcollection defines cell values for the row. Values must be text elements containing the formulas to assign to the cells.

If the formula is a string, use the period `.` as numerical separator and the comma `,` as parameter separator. If a 4D method is used, it must be allowed with the [VP SET ALLOWED METHODS](#) command.

You remove the formulas in *rangeObj* by replacing them with an empty string ("").

## Example 1

```
$formulas:=New collection
$formulas.push(New collection("MAX(B11,C11,D11)";"myMethod(G4)")) // 
First row
$formulas.push(New collection("SUM(B11:D11)";"AVERAGE(B11:D11)")) // 
Second row

VP SET FORMULAS(VP Cell("ViewProArea";6;3);$formulas) // Set the cells
with the formulas

myMethod:
$0:=$1*3.33
```

## Example 2

To remove formulas:

```
$formulas:=New collection
$formulas.push(New collection("");"")) // first collection
$formulas.push(New collection("");"")) // second collection

VP SET FORMULAS(VP Cell("ViewProArea";0;0);$formulas) // Assign to
cells
```

## See also

[VP Get Formulas](#)  
[VP GET VALUES](#)  
[VP SET FORMULA](#)  
[VP SET VALUES](#)

## VP SET FROZEN PANES

**VP SET FROZEN PANES** ( *vpAreaName* : Text ; *paneObj* : Object { ; *sheet* : Integer } )

Parameter	Type	Description
<i>vpAreaName</i>	Text -> 4D View Pro area form object name	
<i>paneObj</i>	Object -> Object containing frozen column and row information	
<i>sheet</i>	Integer -> Sheet index (current sheet if omitted)	

### Description

The `VP SET FROZEN PANES` command sets the frozen status of the columns and rows in the *paneObj* so they are always displayed in the *vpAreaName*. Frozen columns and rows are fixed in place and do not move when the rest of the document is scrolled. A solid line is displayed to indicate that columns and rows are frozen. The location of the line depends on where the frozen column or row is on the sheet:

- **Columns on the left or right:** For columns on the left of the sheet, the line is displayed on the right side of the last frozen column. For columns on the right side of the sheet, the line is displayed on the left side of the first frozen column.
- **Rows on the top or bottom:** For rows at the top of the sheet, the line is displayed below the last frozen row. For rows at the bottom of the sheet, the line is displayed above the first frozen row.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

You can pass an object defining the columns and rows to freeze in the *paneObj* parameter. Setting the value of any of the column or row properties equal to zero resets (unfreezes) the property. If a property is set to less than zero, the command does nothing. You can pass:

### Property Type Description

columnCount	Integer	The number of frozen columns on the left of the sheet
trailingColumnCount	Integer	The number of frozen columns on the right of the sheet
rowCount	Integer	The number of frozen rows on the top of the sheet
trailingRowCount	Integer	The number of frozen rows on the bottom of the sheet

In the optional *sheet* parameter, you can designate a specific spreadsheet where the range will be defined (counting begins at 0). If omitted, the current spreadsheet is used by default. You can explicitly select the current spreadsheet with the following constant:

- `vk current sheet`

### Example

You want to freeze the first three columns on the left, two columns on the right, and the first row:

```

C_OBJECT($panes)

$panes:=New object
$panes.columnCount:=3
$panes.trailingColumnCount:=2
$panes.rowCount:=1

VP SET FROZEN PANES ("ViewProArea";$panes)

```

## See also

[VP Get frozen panes](#)

## VP SET NUM VALUE

**VP SET NUM VALUE** ( *rangeObj* : Object ; *numberValue* : Number { ; *formatPattern* : Text } )

Parameter	Type	Description
<i>rangeObj</i>	Object	-> Range object
<i>numberValue</i>	Number	-> Number value to set
<i>formatPattern</i>	Text	-> Format of value

## Description

The `VP SET NUM VALUE` command assigns a specified numeric value to a designated cell range.

In *rangeObj*, pass a range of the cell(s) (created for example with [VP Cell](#) or [VP Column](#)) whose value you want to specify. If *rangeObj* includes multiple cells, the value specified will be repeated in each cell.

The *numberValue* parameter specifies a numeric value to be assigned to the *rangeObj*.

The optional *formatPattern* defines a [pattern](#) for the *numberValue* parameter.

## Example

```

//Set the cell value to 2
VP SET NUM VALUE(VP Cell("ViewProArea";3;2);2)

//Set the cell value and format it in dollars
VP SET NUM VALUE(VP Cell("ViewProArea";3;2);12.356;"_($* #,##0.00_)")

```

## See also

[Cell format](#)

[VP SET VALUE](#)

## VP SET PRINT INFO

**VP SET PRINT INFO** ( *vpAreaName* : Text ; *printInfo* : Object { ; *sheet* : Integer } )

Parameter	Type	Description
vpAreaNameText	->4D View Pro area name	
printInfo	Object	->Object containing printing attributes
sheet	Integer->	Sheet index (current sheet if omitted)

## Description

The `VP SET PRINT INFO` command defines the attributes to use when printing the *vpAreaName*.

Pass the name of the 4D View Pro area to print in *vpAreaName*. If you pass a name that does not exist, an error is returned.

You can pass an object containing definitions for various printing attributes in the *printInfo* parameter. To view the full list of the available attributes, see [Print Attributes](#).

In the optional *sheet* parameter, you can designate a specific spreadsheet to print (counting begins at 0). If omitted, the current spreadsheet is used by default. You can explicitly select the current spreadsheet with the following constant:

- `vk current sheet`

## Example

The following code will print a 4D View Pro area to a PDF document:

```
var $printInfo : Object

//declare print attributes object
$printInfo:=New object

//define print attributes
$printInfo.headerCenter:="&BS.H.I.E.L.D. &A Sales Per Region"
$printInfo.firstPageNumber:=1
$printInfo.footerRight:="page &P of &N"
$printInfo.orientation:=vk print page orientation landscape
$printInfo.centering:=vk print centering horizontal
$printInfo.columnStart:=0
$printInfo.columnEnd:=8
$printInfo.rowStart:=0
$printInfo.rowEnd:=24

$printInfo.showGridLine:=True

//Add corporate logo
$printInfo.headerLeftImage:=logo.png
$printInfo.headerLeft:="&G"

$printInfo.showRowHeader:=vk print visibility hide
$printInfo.showColumnHeader:=vk print visibility hide
$printInfo.fitPagesWide:=1
$printInfo.fitPagesTall:=1

//print PDF document
VP SET PRINT INFO ("ViewProArea";$printInfo)

//export the PDF
VP EXPORT DOCUMENT("ViewProArea";"Sales2018.pdf";New
object("formula";Formula(ALERT("PDF ready!"))))
```

The PDF:

## See also

[4D View Pro print attributes](#)

[VP Convert to picture](#)

[VP Get print info](#)

[VP PRINT](#)

## VP SET ROW ATTRIBUTES

**VP SET ROW ATTRIBUTES** ( *rangeObj* : Object ; *propertyObj* : Object )

Parameter	Type	Description
<i>rangeObj</i>	Object->Range of rows	
<i>propertyObj</i>	Object->Object containing row properties	

### Description

The `VP SET ROW ATTRIBUTES` command applies the attributes defined in the *propertyObj* to the rows in the *rangeObj*.

In the *rangeObj*, pass an object containing a range. If the range contains both columns and rows, attributes are applied only to the rows.

The *propertyObj* parameter lets you specify the attributes to apply to the rows in the *rangeObj*. These attributes are:

Property	Type	Description
<i>height</i>	number	Row height expressed in pixels
<i>pageBreak</i>	boolean	True to insert a page break before the first row of the range, else false
<i>visible</i>	boolean	True if the row is visible, else false
<i>resizable</i>	boolean	True if the row can be resized, else false
<i>header</i>	text	Row header text

### Example

You want to change the size of the second row and set the header:

```
var $row; $properties : Object  
  
$row:=VP Row("ViewProArea";1)  
$properties:=New object("height";75;"header";"June")  
  
VP SET ROW ATTRIBUTES($row;$properties)
```

## See also

[VP Get row attributes](#)

[VP get column attributes](#)

[VP SET ROW ATTRIBUTES](#)

## VP SET ROW COUNT

**VP SET ROW COUNT** ( *vpAreaName* : Text ; *rowCount* : Integer { ; *sheet* : Integer } )

Parameter	Type	Description
<i>vpAreaName</i>	Text -> 4D View Pro area form object name	
<i>rowCount</i>	Integer-> Number of rows	
<i>sheet</i>	Integer-> Sheet index (current sheet if omitted)	

### Description

The `VP SET ROW COUNT` command defines the total number of rows in *vpAreaName*.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

Pass the total number of rows in the *rowCount* parameter. *rowCount* must be greater than 0.

In the optional *sheet* parameter, you can designate a specific spreadsheet where the *rowCount* will be applied (counting begins at 0). If omitted, the current spreadsheet is used by default. You can explicitly select the current spreadsheet with the following constant:

- `vk current sheet`

### Example

The following code defines five rows in the 4D View Pro area:

```
VP SET ROW COUNT("ViewProArea";5)
```

### See also

[VP Get column count](#)  
[VP get row-count](#)  
[VP SET COLUMN COUNT](#)

## VP SET SELECTION

**VP SET SELECTION** ( *rangeObj* : Object )

### Parameter Type Description

|*rangeObj* |Object|->|Range object of cells|

### Description

The `VP SET SELECTION` command defines the specified cells as the selection and the first cell as the active cell.

In *rangeObj*, pass a range object of cells to designate as the current selection.

### Example

```
$currentSelection:=VP Combine ranges (VP Cells ("myVPArea";3;2;1;6);VP Cells ("myVPArea";5;7;1;7))
VP SET SELECTION($currentSelection)
```

## See also

[VP Get active cell](#)  
[VP Get selection](#)  
[VP RESET SELECTION](#)  
[VP SET ACTIVE CELL](#)  
[VP ADD SELECTION](#)  
[VP SHOW CELL](#)

## VP SET SHEET COUNT

**VP SET SHEET COUNT** ( *vpAreaName* : Text ; *number* : Integer )

Parameter	Type	Description
<i>vpAreaName</i>	Text	-> 4D View Pro area form object name
<i>number</i>	Integer	-> Number of sheets

## Description

The `VP SET SHEET COUNT` command sets the number of sheets in *vpAreaName*.

In *number*, pass a number corresponding to how many sheets the document will contain after the command is executed.

**Warning:** The command will delete sheets if the previous amount of sheets in your document is superior to the number passed. For example, if there are 5 sheets in your document and you set the sheet count to 3, the command will delete sheets number 4 and 5.

## Example

The document currently has one sheet:



To set the number of sheets to 3:

```
VP SET SHEET COUNT("ViewProArea";3)
```



## See also

[VP Get sheet count](#)

## VP SET SHEET NAME

**VP SET SHEET NAME** ( *vpAreaName* : Text ; *name* : Text {; *sheet*: Integer} )

Parameter	Type	Description
vpAreaName	Text -> 4D View Pro area name	4D View Pro area form object
name	Text -> New name for the sheet	
sheet Integer	-> Index of the sheet to be renamed	

## Description

The `VP SET SHEET NAME` command renames a sheet in the document loaded in *vpAreaName*.

In *vpAreaName*, pass the name of the 4D View Pro area.

In *name*, pass a new name for the sheet.

In *sheet*, pass the index of the sheet to rename.

Indexing starts at 0.

If no *index* is passed, the command renames the current sheet.

The new name cannot contain the following characters: `* , : , [ , ] , ? , \ , /`

The command does nothing if:

- the new name contains forbidden characters
- the new name's value is blank
- the new name already exists
- the passed *index* does not exist

## Example

Set the third sheet's name to "Total first quarter":

```
VP SET SHEET NAME ("ViewProArea"; "Total first quarter"; 2)
```



## VP SET SHEET OPTIONS

**VP SET SHEET OPTIONS** ( *vpAreaName* : Text; *sheetOptions* : Object { ; *sheet* : Integer } )

Parameter	Type	Description
vpAreaName	Object -> 4D View Pro area name	
sheetOptions	Object -> Sheet option(s) to set	
sheet	Object ->	Sheet index (current sheet if omitted)

## Description

The `VP SET SHEET OPTIONS` command allows defining various sheet options of the *vpAreaName* area.

Pass the name of the 4D View Pro area in *vpAreaName*. If you pass a name that does not exist, an error is returned.

Pass an object containing definitions for the options to set in the *sheetOptions* parameter. To view the full list of the available options, see the [Sheet Options](#) paragraph.

In the optional *sheet* parameter, you can designate a specific spreadsheet (counting begins at 0). If omitted, the current spreadsheet is used by default. You can explicitly select the current spreadsheet with the following constant:

- `vk current sheet`

### Example 1

You want to protect all cells except the range C5:D10:

```
// Activate protection on the current sheet
var $options : Object

$options:=New object
$options.isProtected:=True
VP SET SHEET OPTIONS("ViewProArea";$options)

// mark cells C5:D10 as 'unlocked'
VP SET CELL STYLE(VP Cells("ViewProArea";2;4;2;6);New
object("locked";False))
```

### Example 2

You need to protect your document while your users can resize rows and columns:

```
var $options : Object

$options:=New object
// Activate protection
$options.isProtected:=True
$options.protectionOptions:=New object
// Allow user to resize rows
$options.protectionOptions.allowResizeRows=True;
// Allow user to resize columns
$options.protectionOptions.allowResizeColumns=True;

// Apply protection on the current sheet
VP SET SHEET OPTIONS("ViewProArea";$options)
```

### Example 3

You want to customize the colors of your sheet tabs, frozen lines, grid lines, selection background and selection border:

```

var $options : Object

$options:=New object
// Customize color of Sheet 1 tab
$options.sheetTabColor:="Black"
$options.gridline:=New object("color";"Purple")
$options.selectionBackColor:="rgb(255,128,0,0.4)"
$options.selectionBorderColor:="Yellow"
$options.frozenlineColor:="Gold"

VP SET SHEET OPTIONS("ViewProArea";$options;0)

// Customize color of Sheet 2 tab
$options.sheetTabColor:="red"

VP SET SHEET OPTIONS("ViewProArea";$options;1)

// Customize color of Sheet 3 tab
$options.sheetTabColor:="blue"

VP SET SHEET OPTIONS("ViewProArea";$options;2)

```

**Result:**

#### Example 4

You want to hide the grid lines as well as the row and column headers.

```

var $options : Object

$options:=New object
$options.gridline:=New object()
$options.gridline.showVerticalGridline:=False
$options.gridline.showHorizontalGridline:=False
$options.rowHeaderVisible:=False
$options.colHeaderVisible:=False

VP SET SHEET OPTIONS("ViewProArea";$options)

```

**Result:**

#### See also

[4D View Pro sheet options](#)  
[VP Get sheet options](#)

### VP SET SHOW PRINT LINES

**VP SET SHOW PRINT LINES** ( *vpAreaName* : Text {; *visible* : Boolean} {; *sheet* : Integer} )

Parameter	Type	Description
<i>vpAreaName</i>	Text	->4D View Pro area form object name
<i>visible</i>	Boolean	-> Print lines displayed if True (default), hidden if False
<i>sheet</i>	Integer	-> Sheet index (current sheet if omitted)

#### Description

The `VP SET SHOW PRINT LINES` command sets whether to display print preview lines in a spreadsheet..

In *vpAreaName*, pass the name of the 4D View Pro area.

In *visible*, pass `True` to display the print lines, and `False` to hide them. `True` is passed by default.

In *sheet*, pass the index of the target sheet. If no index is specified, the command applies to the current sheet.

Indexing starts at 0.

The position of a spreadsheet's print lines varies according to that spreadsheet's page breaks.

## Example

The following code displays print lines in a document's second sheet:

```
VP SET SHOW PRINT LINES ("ViewProArea";True;1)
```

With a page break:

## See also

[4D Get show print lines](#)

## VP SET TABLE COLUMN ATTRIBUTES

- History

**VP SET TABLE COLUMN ATTRIBUTES** ( *vpAreaName* : Text ; *tableName* : Text ; *column* : Integer ; *attributes* : Object {; *sheet* : Integer } )

Parameter	Type	Description
<i>vpAreaName</i>	Text	->4D View Pro area form object name
<i>tableName</i>	Text	->Table name
<i>column</i>	Integer	->Index of the column in the table
<i>attributes</i>	Object	->Attribute(s) to apply to the <i>column</i>
<i>sheet</i>	Integer	->Sheet index (current sheet if omitted)

## Description

The `VP SET TABLE COLUMN ATTRIBUTES` command applies the defined *attributes* to the *column* in the *tableName*.

In *vpAreaName*, pass the name of the 4D View Pro area.

In the *attributes* parameter, pass an object that contains the properties to set:

Property	Type	Description
dataField	text	Table column's property name in the data context.
name	text	Table column's name. Must be unique in the table. If this name already used by another column, it is not applied and a default name is automatically used.
formula	text	Sets the formula for each column cell. See <a href="#">Structured Reference Formulas in the SpreadJS documentation</a>
footerText	text	Column footer value.
footerFormula	text	Column footer formula.
filterButtonVisible	boolean	Sets whether the table column's filter button is displayed (default is <code>True</code> when the table is created).

In *sheet*, pass the index of the target sheet. If no index is specified or if you pass -1, the command applies to the current sheet.

Indexing starts at 0.

If *tableName* is not found or if *column* is higher than the number of columns, the command does nothing.

## Example

You create a table with a data context:

```
var $context; $options : Object

$context:=New object()
$context.col:=New collection()
$context.col.push(New object("name"; "Smith"; "firstname"; "John";
"salary"; 10000))
$context.col.push(New object("name"; "Wesson"; "firstname"; "Jim";
"salary"; 50000))
$context.col.push(New object("name"; "Gross"; "firstname"; "Maria";
"salary"; 10500))
VP SET DATA CONTEXT("ViewProArea"; $context)

    //Define the columns for the table
$options:=New object()
$options.tableColumns:=New collection()
$options.tableColumns.push(New object("name"; "Last Name"; "dataField";
"name"))
$options.tableColumns.push(New object("name"; "Salary"; "dataField";
"salary"))

VP CREATE TABLE(VP Cells("ViewProArea"; 1; 1; 2; 3); "PeopleTable";
"col"; $options)
```

	A	B	C
1			
2	Last Name ▾	Salary ▾	
3	Smith	10000	
4	Wesson	50000	
5	Gross	10500	

Then you want to insert a column with data from the data context and hide some filter buttons:

```

    //insert a column
VP INSERT TABLE COLUMNS("ViewProArea"; "PeopleTable"; 1; 1)

var $param : Object
$param:=New object()
    // Bind the column to the firstname field from the datacontext
$param.dataField:="firstname"
    // Change the default name of the column to "First name"
    // and hide the filter button
$param.name:="First Name"
$param.filterButtonVisible:=False

VP SET TABLE COLUMN ATTRIBUTES("ViewProArea"; "PeopleTable"; 1; $param)

    // Hide the filter button of the first column
VP SET TABLE COLUMN ATTRIBUTES("ViewProArea"; "PeopleTable"; 0; \
    New object("filterButtonVisible"; False))

```

	A	B	C	D
1				
2	Last Name	FirstName	Salary ▾	
3	Smith	John	10000	
4	Wesson	Jim	50000	
5	Gross	Maria	10500	

## See also

[VP CREATE TABLE](#)

[VP Find table](#)

[VP Get table column attributes](#)

[VP RESIZE TABLE](#)

## VP SET TABLE THEME

- History

**VP SET TABLE THEME** ( *vpAreaName* : Text ; *tableName* : Text ; *options* : cs.ViewPro.TableTheme )

Parameter	Type	Description
<i>vpAreaName</i> Text	-> 4D View Pro area form object name	
<i>tableName</i> Text	-> Table name	
<i>options</i> cs.ViewPro.TableTheme	-> Table theme properties to modify	

## Description

The `VP SET TABLE THEME` command modifies the current theme of the *tableName*.

In *vpAreaName*, pass the name of the 4D View Pro area and in *tableName*, the name of the table to modify.

In the *options* parameter, pass an object of the [cs.ViewPro.TableTheme class](#) that contains the theme properties to modify.

## Example 1

You want to set a predefined theme to a table:

```

var $param : cs.ViewPro.TableTheme
$param:=cs.ViewPro.TableTheme.new()
$param.theme:="medium2"
VP SET TABLE THEME("ViewProArea"; "myTable"; $param)

```

## Example 2

You want to have this alternate column rendering:

```

var $param : cs.ViewPro.TableTheme
$param:=cs.ViewPro.TableTheme.new()

// Enable the band column rendering
$param.bandColumns:=True
$param.bandRows:=False

// Create the theme object with header and column styles
$param.theme:=cs.ViewPro.TableThemeOptions.new()

var $styleHeader; $styleColumn; $styleColumn2 : cs.ViewPro.TableStyle

$styleHeader:=cs.ViewPro.TableStyle.new()
$styleHeader.backColor:="Gold"
$styleHeader.foreColor:="#03045E"
$param.theme.headerRowStyle:=$styleHeader

$styleColumn1:=cs.ViewPro.TableStyle.new()
$styleColumn1.backColor:="SkyBlue"
$styleColumn1.foreColor:="#03045E"
$param.theme.firstColumnStripStyle:=$styleColumn1

$styleColumn2:=cs.ViewPro.TableStyle.new()
$styleColumn2.backColor:="LightCyan"
$styleColumn2.foreColor:="#03045E"
$param.theme.secondColumnStripStyle:=$styleColumn2

VP SET TABLE THEME("ViewProArea"; "myTable"; $param)

```

## See also

[VP CREATE TABLE](#)  
[VP Get table theme](#)

## VP SET TEXT VALUE

**VP SET TEXT VALUE** ( *rangeObj* : Object ; *textValue* : Text { ; *formatPattern* : Text } )

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>rangeObj</i>	Object -> Range object	
<i>textValue</i>	Text ->	Text value to set
<i>formatPattern</i>	Text	-> Format of value

## Description

The `VP SET TEXT VALUE` command assigns a specified text value to a designated cell range.

In *rangeObj*, pass a range of the cell(s) (created for example with [VP\\_Cell](#) or [VP\\_Column](#)) whose value you want to specify. If *rangeObj* includes multiple cells, the value specified will be repeated in each cell.

The *textValue* parameter specifies a text value to be assigned to the *rangeObj*.

The optional *formatPattern* defines a [pattern](#) for the *textValue* parameter.

## Example

```
VP SET TEXT VALUE(VP Cell("ViewProArea";3;2);"Test 4D View Pro")
```

## See also

[Cell Format](#)

[VP\\_SET\\_VALUE](#)

## VP\_SET\_TIME\_VALUE

**VP\_SET\_TIME\_VALUE** (*rangeObj* : Object ; *timeValue* : Text { ; *formatPattern* : Text } )

Parameter	Type	Description
<i>rangeObj</i>	Object-> Range object	
<i>timeValue</i>	Text -> Time value to set	
<i>formatPattern</i>	Text -> Format of value	

## Description

The `VP_SET_TIME_VALUE` command assigns a specified time value to a designated cell range.

In *rangeObj*, pass a range of the cell(s) (created for example with [VP\\_Cell](#) or [VP\\_Column](#)) whose value you want to specify. If *rangeObj* includes multiple cells, the value specified will be repeated in each cell.

The *timeValue* parameter specifies a time expressed in seconds to be assigned to the *rangeObj*.

The optional *formatPattern* defines a [pattern](#) for the *timeValue* parameter.

## Example

```
//Set the value to the current time  
VP SET TIME VALUE(VP Cell("ViewProArea";5;2);Current time)  
  
//Set the value to a specific time with a designated format  
VP SET TIME VALUE(VP Cell("ViewProArea";5;2);?12:15:06?;vk pattern long time)
```

## See also

[Cell Format](#)

[VP\\_SET\\_DATE\\_TIME\\_VALUE](#)

[VP\\_SET\\_VALUE](#)

## VP\_SET\_VALUE

## **VP SET VALUE** ( *rangeObj* : Object ; *valueObj* : Object )

Parameter	Type	Description
<i>rangeObj</i>	Object-> Range object	
<i>valueObj</i>	Object->	Cell values and format options

### Description

The `VP SET VALUE` command assigns a specified value to a designated cell range.

The command allows you to use a generic code to set and format the types of values in *rangeObj*, whereas other commands, such as [VP SET TEXT VALUE](#) and [VP SET NUM VALUE](#), reduce the values to specific types.

In *rangeObj*, pass a range of the cell(s) (created for example with [VP Cell](#) or [VP Column](#)) whose value you want to specify. If *rangeObj* includes multiple cells, the value specified will be repeated in each cell.

The parameter *valueObj* is an object that includes properties for the value and the [format](#) to assign to *rangeObj*. It can include the following properties :

Property	Type	Description
<i>value</i>	Integer, Real, Boolean, Text, Date, Null	Value to assign to <i>rangeObj</i> (except- time). Pass null to erase the content of the cell.
<i>time</i>	Real	Time value (in seconds) to assign to <i>rangeObj</i>
<i>format</i>	Text	Pattern for value/time property. For information on patterns and formatting characters, please refer to the <a href="#">Cell Format</a> paragraph.

### Example

```

//Set the cell value as False
VP SET VALUE(VP Cell("ViewProArea";3;2);New object("value";False))

//Set the cell value as 2
VP SET VALUE(VP Cell("ViewProArea";3;2);New object("value";2))

//Set the cell value as $125,571.35
VP SET VALUE(VP Cell("ViewProArea";3;2);New
object("value";125571.35;"format";"_($* #,##0.00_)"))

//Set the cell value as Hello World!
VP SET VALUE(VP Cell("ViewProArea";3;2);New object("value";"Hello
World!"))

//Set the cell value as current date
VP SET VALUE(VP Cell("ViewProArea";4;2);New object("value";Current
date))

//Set the cell value as current hour
VP SET VALUE(VP Cell("ViewProArea";5;2);New object("time";Current
hour))

//Set the cell value as specific date and time
VP SET VALUE(VP Cell("ViewProArea";3;9);New object("value";!2024-12-
18!); "time";?14:30:10?;"format";vk pattern full date time))

//Erase cell content
VP SET VALUE(VP Cell("ViewProArea";3;9);New object("value";Null))

```

## See also

[Cell Format](#)  
[VP Get values](#)  
[VP SET VALUE](#)  
[VP SET BOOLEAN VALUE](#)  
[VP SET DATE TIME VALUE](#)  
[VP SET FIELD](#)  
[VP SET FORMULA](#)  
[VP SET NUM VALUE](#)  
[VP SET TEXT VALUE](#)  
[VP SET TIME VALUE](#)

## VP SET VALUES

**VP SET VALUES** ( *rangeObj* : Object ; *valuesCol* : Collection )

Parameter	Type	Description
<i>rangeObj</i>	Object	-> Range object
<i>valuesCol</i>	Collection	-> Collection of values

### Description

The `VP SET VALUES` command assigns a collection of values starting at the specified cell range.

In *rangeObj*, pass a range for the cell (created with [VP Cell](#)) whose value you want to specify. The cell defined in the *rangeObj* is used to determine the starting point.

- If *rangeObj* is not a cell range, only the first cell of the range is used.
- If *rangeObj* includes multiple ranges, only the first cell of the first range is used.

The *valuesCol* parameter is two-dimensional:

- The first-level collection contains subcollections of values. Each subcollection defines a row. Pass an empty collection to skip a row.
- Each subcollection defines cell values for the row. Values can be Integer, Real, Boolean, Text, Date, Null, or Object. If the value is an object, it can have the following properties:

Property	Type	Description
value	Integer, Real, Boolean, Text, Date, Null	Value in the cell (except- time)
time	Real	Time value (in seconds)

## Example

```
$param:=New collection
$param.push(New collection(1;2;3;False)) //first row, 4 values
$param.push(New collection) //second row, untouched
$param.push(New collection(4;5;Null;"hello";"world")) // third row, 5
values
$param.push(New collection(6;7;8;9)) // fourth row, 4 values
$param.push(New collection(Null;New object("value";Current
date;"time";42))) //fifth row, 1 value

VP SET VALUES(VP Cell("ViewProArea";2;1);$param)
```

## See also

[VP Get formulas](#)  
[VP Get value](#)  
[VP Get Values](#)  
[VP SET FORMULAS](#)  
[VP SET VALUE](#)

## VP SET WORKBOOK OPTIONS

**VP SET WORKBOOK OPTIONS** (*vpAreaName* : Text ; *optionObj* : Object)

Parameter	Type	Description
<i>vpAreaName</i>	Text	-> 4D View Pro area form object name
<i>optionObj</i>	Object	-> Object containing the workbook options to be set

## Description

**VP SET WORKBOOK OPTIONS** sets the workbook options in *vpAreaName*.

In *vpAreaName*, pass the name of the 4D View Pro area.

In *optionObj*, pass the workbook options to apply to *vpAreaName*.

If *optionObj* is empty, the command does nothing.

Modified workbook options are saved with the document.

The following table lists the available workbook options:

Property	Type	Description										
allowUserDragMerge	boolean	The drag merge operation is allowed (select cells and drag the selection to merge cells)										
allowAutoCreateHyperlink	boolean	Enables automatic creation of hyperlinks in the spreadsheet.										
allowContextMenu	boolean	The built-in context menu can be opened.										
allowCopyPasteExcelStyle	boolean	Styles from a spreadsheet can be copied and pasted to Excel, and vice-versa.										
allowDynamicArray	boolean	Enables dynamic arrays in worksheets										
allowExtendPasteRange	boolean	Extends the pasted range if the pasted range is not enough for the pasted data										
allowSheetReorder	boolean	Sheet reordering is allowed										
allowUndo	boolean	Undoing edits is allowed.										
allowUserDeselect	boolean	Deselecting specific cells from a selection is allowed.										
allowUserDragDrop	boolean	Drag and drop of range data is allowed										
allowUserDragFill	boolean	Drag fill is allowed										
allowUserEditFormula	boolean	Formulas can be entered in cells										
allowUserResize	boolean	Columns and rows can be resized										
allowUserZoom	boolean	Zooming (ctrl + mouse wheel) is allowed										
autoFitType	number	Content is formatted to fit in cells, or cells and headers. Available values:										
		<table border="1"> <thead> <tr> <th>Constant</th><th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>vk auto fit type cell</td><td>0</td><td>The content autofits cells</td></tr> <tr> <td>vk auto fit type cell with header</td><td>1</td><td>The content autofits cells and headers</td></tr> </tbody> </table>	Constant	Value	Description	vk auto fit type cell	0	The content autofits cells	vk auto fit type cell with header	1	The content autofits cells and headers	
Constant	Value	Description										
vk auto fit type cell	0	The content autofits cells										
vk auto fit type cell with header	1	The content autofits cells and headers										
backColor	string	A color string used to represent the background color of the area, such as "red", "#FFFF00", "rgb(255,0,0)", "Accent 5". The initial backgroundcolor is hidden when a backgroundImage is set.										
backgroundImage	string / picture / file	Background image for the area.										
backgroundImageLayout	number	How the background image is displayed. Available values:										
		<table border="1"> <thead> <tr> <th>Constant</th><th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>vk image center</td><td>1</td><td>In the center of the area.</td></tr> <tr> <td>vk image none</td><td>3</td><td>In the upper left corner of the area with its original size.</td></tr> <tr> <td>vk image stretch</td><td>0</td><td>Fills the area.</td></tr> </tbody> </table>	Constant	Value	Description	vk image center	1	In the center of the area.	vk image none	3	In the upper left corner of the area with its original size.	vk image stretch
Constant	Value	Description										
vk image center	1	In the center of the area.										
vk image none	3	In the upper left corner of the area with its original size.										
vk image stretch	0	Fills the area.										

Property	Type	Constant Value	Description
calcOnDemand	boolean		Formulas are calculated only when they are demanded.
columnResizeMode	number		Resize mode for columns. Available values:
		<b>Constant Value</b>	<b>Description</b>
		vk resize mode	Use normal resize mode (i.e remaining columns are affected)
		normal	
		vk resize mode	Use split mode (i.e remaining columns are not affected)
		split	
			Headers to include when data is copied to or pasted. Available values:
		<b>Constant Value</b>	<b>Description</b>
		vk copy paste header options all headers	Includes selected headers when data is copied; overwrites selected headers when data is pasted.
		vk copy paste header options column headers	Includes selected column headers when data is copied; overwrites selected column headers when data is pasted.
copyPasteHeaderOptions	number	vk copy paste header options no headers	Column and row headers are not included when data is copied; does not overwrite selected column or row headers when data is pasted.
		vk copy paste header options row headers	Includes selected row headers when data is copied; overwrites selected row headers when data is pasted.
customList	collection		The list for users to customize drag fill, prioritize matching this list in each fill. Each collection item is a collection of strings. See on <a href="#">GrapeCity's website</a> .
cutCopyIndicatorBorderColor	string		Border color for the indicator displayed when the user cuts or copies the selection.
cutCopyIndicatorVisible	boolean		Display an indicator when copying or cutting the selected item.
			The default drag fill type. Available values :
		<b>Constant Value</b>	<b>Description</b>

Property	Type	Description						
defaultDragFillType	number	<p>vk auto fill type auto vk auto fill type clear values vk auto fill type copycells vk auto fill type fill formatting only vk auto fill type fill series vk auto fill type fill without formatting</p> <p>Automatically fills cells.</p> <p>Clears cell values.</p> <p>Fills cells with all data objects, including values, formatting, and formulas.</p> <p>Fills cells only with formatting.</p> <p>Fills cells with series.</p> <p>Fills cells with values and not formatting.</p>						
enableAccessibility	boolean	Accessibility support is enabled in the spreadsheet.						
enableFormulaTextbox	boolean	The formula text box is enabled.						
grayAreaBackColor	string	A color string used to represent the background color of the gray area , such as "red", "#FFFF00", "rgb(255,0,0)", "Accent 5", and so on.						
highlightInvalidData	boolean	Invalid data is highlighted.						
iterativeCalculation	boolean	Enables iterative calculation. See on <a href="#">GrapeCity's website</a> .						
iterativeCalculationMaximumChange	numeric	Maximum amount of change between two calculation values.						
iterativeCalculationMaximumIterations	numeric	Number of times the formula should recalculate.						
newTabVisible	boolean	Display a special tab to let users insert new sheets.						
numbersFitMode	number	<p>Changes display mode when date/number data width is longer than column width. Available values:</p> <table> <thead> <tr> <th>Constant Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>vk numbers fit mode mask</td><td>Replace data content with "###" and shows tip</td></tr> <tr> <td>vk numbers fit mode overflow</td><td>Display data content as a string. If next cell is empty, overflow the content.</td></tr> </tbody> </table> <p>Paste or skip pasting data in invisible ranges:</p> <ul style="list-style-type: none"> <li>• False (default): paste data</li> <li>• True: Skip pasting in invisible ranges</li> </ul>	Constant Value	Description	vk numbers fit mode mask	Replace data content with "###" and shows tip	vk numbers fit mode overflow	Display data content as a string. If next cell is empty, overflow the content.
Constant Value	Description							
vk numbers fit mode mask	Replace data content with "###" and shows tip							
vk numbers fit mode overflow	Display data content as a string. If next cell is empty, overflow the content.							
pasteSkipInvisibleRange	boolean							

Property	Type	See <a href="#">GrapeCity Document</a> for more information on invisible ranges.									
referenceStyle	number	Style for cell and range references in cell formulas. Available values:									
		<table> <thead> <tr> <th>Constant</th><th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>vk reference style A1</td><td>0</td><td>Use A1 style.</td></tr> <tr> <td>vk reference style R1C1</td><td>1</td><td>Use R1C1 style</td></tr> </tbody> </table>	Constant	Value	Description	vk reference style A1	0	Use A1 style.	vk reference style R1C1	1	Use R1C1 style
Constant	Value	Description									
vk reference style A1	0	Use A1 style.									
vk reference style R1C1	1	Use R1C1 style									
resizeZeroIndicator	number	Drawing policy when the row or column is resized to zero. Available values:									
		<table> <thead> <tr> <th>Constant</th><th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>vk resize zero indicator default</td><td>0</td><td>Uses the current drawing policy when the row or column is resized to zero.</td></tr> <tr> <td>vk resize zero indicator enhanced</td><td>1</td><td>Draws two short lines when the row or column is resized to zero.</td></tr> </tbody> </table>	Constant	Value	Description	vk resize zero indicator default	0	Uses the current drawing policy when the row or column is resized to zero.	vk resize zero indicator enhanced	1	Draws two short lines when the row or column is resized to zero.
Constant	Value	Description									
vk resize zero indicator default	0	Uses the current drawing policy when the row or column is resized to zero.									
vk resize zero indicator enhanced	1	Draws two short lines when the row or column is resized to zero.									
rowResizeMode	number	The way rows are resized. Available values are the same as columnResizeMode									
scrollbarAppearance	number	Scrollbar appearance. Available values:									
		<table> <thead> <tr> <th>Constant</th><th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>vk scrollbar appearance mobile</td><td>1</td><td>Mobile scrollbar appearance.</td></tr> <tr> <td>vk scrollbar appearance skin (default)</td><td>0</td><td>Excel-like classic scrollbar appearance.</td></tr> </tbody> </table>	Constant	Value	Description	vk scrollbar appearance mobile	1	Mobile scrollbar appearance.	vk scrollbar appearance skin (default)	0	Excel-like classic scrollbar appearance.
Constant	Value	Description									
vk scrollbar appearance mobile	1	Mobile scrollbar appearance.									
vk scrollbar appearance skin (default)	0	Excel-like classic scrollbar appearance.									
scrollbarMaxAlign	boolean	The scroll bar aligns with the last row and column of the active sheet.									
scrollbarShowMax	boolean	The displayed scroll bars are based on the entire number of columns and rows in the sheet.									
scrollByPixel	boolean	Enable precision scrolling by pixel.									
scrollIgnoreHidden	boolean	The scroll bar ignores hidden rows or columns.									
scrollPixel	integer	Decides scrolling by that number of pixels at a time when scrollByPixel is true. The final scrolling pixels are the result of <code>scrolling delta * scrollPixel</code> . For example: scrolling delta is 3, scrollPixel is 5, the final scrolling pixels are 15.									
showDragDropTip	boolean	Display the drag-drop tip.									
showDragFillSmartTag	boolean	Display the drag fill dialog.									
showDragFillTip	boolean	Display the drag-fill tip.									
showHorizontalScrollbar	boolean	Display the horizontal scroll bar.									
		How to display the resize tip. Available values:									

Property	Type	Constant	Value	Description
showResizeTip	number	vk show resize tip both vk show resize tip column vk show resize tip none vk show resize tip row	3 1 0 2	Horizontal and vertical resize tips are displayed. Only the horizontal resize tip is displayed. No resize tip is displayed. Only the vertical resize tip is displayed.
		How to display the scroll tip. Available values:		
		Constant	Value	Description
		vk show scroll tip both vk show scroll tip horizontal vk show scroll tip none vk show scroll tip vertical	3 1 0 2	Horizontal and vertical scroll tips are displayed. Only the horizontal scroll tip is displayed. No scroll tip is displayed. Only the vertical scroll tip is displayed.
showScrollTip	number	boolean	Display the vertical scroll bar.	
tabEditable	boolean	boolean	The sheet tab strip can be edited.	
tabNavigationVisible	boolean	boolean	Display the sheet tab navigation.	
		Position of the tab strip. Available values:		
tabStripPosition	number	Constant	Value	Description
		vk tab strip position bottom	0	Tab strip position is relative to the bottom of the workbook.
		vk tab strip position left	2	Tab strip position is relative to the left of the workbook.
		vk tab strip position right	3	Tab strip position is relative to the right of the workbook.
		vk tab strip position top	1	Tab strip position is relative to the top of the workbook.
tabStripRatio	number	number	Percentage value (0.x) that specifies how much of the horizontal space will be allocated to the tab strip. The rest of the horizontal area (1 - 0.x) will be allocated to the horizontal scrollbar.	
tabStripVisible	boolean	boolean	Display the sheet tab strip.	
tabStripWidth	number	left or right	Width of the tab strip when position is left or right. Default and minimum is	

Property	Type	Description
useTouchLayout	boolean	Whether to use touch layout to present the Spread component.

## Example

To set the allowExtendpasteRange option in "ViewProArea":

```
var $workbookOptions : Object
$workbookOptions:= New Object
$workbookOptions.allowExtendPasteRange:=True
VP SET WORKBOOK OPTIONS ("ViewProArea";$workbookOptions)
```

## See also

[VP Get workbook options](#)

## VP SHOW CELL

**VP SHOW CELL** ( *rangeObj* : Object { ; *vPos* : Integer; *hPos* : Integer } )

### Parameter Type Description

*rangeObj* Object->Range object

|*vPos* |Integer|->|Vertical view position of cell or row| |*hPos* |Integer|->|Horizontal view position of cell or row|

### Description

The `VP SHOW CELL` command vertically and horizontally repositions the view of the *rangeObj*.

In *rangeObj*, pass a range of cells as an object to designate the cells to be viewed. The view of the *rangeObj* will be positioned vertically or horizontally (i.e., where *rangeObj* appears) based on the *vPos* and *hPos* parameters. The *vPos* parameter defines the desired vertical position to display the *rangeObj*, and the *hPos* parameter defines the desired horizontal position to display the *rangeObj*.

The following selectors are available:

<b>Selector</b>	<b>Description</b>	<b>Available with vPos</b>	<b>Available with hPos</b>
<code>vk position bottom</code>	Vertical alignment to the bottom of cell or row.	X	
<code>vk position center</code>	Alignment to the center. The alignment will be to the cell, row, or column limit according to the view position indicated:	X	X
<code>center</code>	<ul style="list-style-type: none"> <li>• Vertical view position - cell or row</li> <li>• Horizontal view position - cell or column</li> </ul>		
<code>vk position left</code>	Horizontal alignment to the left of the cell or column	X	
<code>vk position nearest row</code>	Alignment to the closest limit (top, bottom, left, right, center). The alignment will be to the cell, row, or column limit according to the view position indicated:		
<code>position</code>	<ul style="list-style-type: none"> <li>• Vertical view position (top, center, bottom) - cell or row</li> <li>• Horizontal view position (left, center, right) - cell or column</li> </ul>	X	X
<code>vk position right</code>	Horizontal alignment to the right of the cell or column	X	
<code>vk position top</code>	Vertical alignment to the top of cell or row	X	

This command is only effective if repositioning the view is possible. For example, if the *rangeObj* is in cell A1 (the first column and the first row) of the current sheet, repositioning the view will make no difference because the vertical and horizontal limits have already been reached (i.e., it is not possible to scroll any higher or any more to the left). The same is true if *rangeObj* is in cell C3 and the view is repositioned to the center or the bottom right. The view remains unaltered.

## Example

You want to view the cell in column AY, row 51 in the center of the 4D View Pro area:

```
$displayCell:=VP Cell("myVPArea";50;50)
// Move the view to show the cell
VP SHOW CELL($displayCell;vk position center;vk position center)
```

Result:

The same code with the vertical and horizontal selectors changed to show the same cell positioned at the top right of the 4D View Pro area:

```
$displayCell:=VP Cell("myVPArea";50;50)
// Move the view to show the cell
VP SHOW CELL($displayCell;vk position top;vk position right)
```

Result:

## See also

[VP ADD CELL](#)  
[VP Get active cell](#)  
[VP Get selection](#)  
[VP RESET SELECTION](#)  
[VP SET ACTIVE CELL](#)  
[VP SET SELECTION](#)

## VP SUSPEND COMPUTING

**VP SUSPEND COMPUTING** ( *vpAreaName* : Text )

Parameter	Type	Description
<i>vpAreaName</i>	Text	4D View Pro area form object name

### Description

The `VP SUSPEND COMPUTING` command stops the calculation of all formulas in *vpAreaName*. This command is useful when you want to suspend calculations in this 4D View Pro area so you can manually make modifications to formulas without encountering errors before you've finished making the changes.

The command pauses the calculation service in 4D View Pro. Formulas that have already been calculated remain unchanged, however any formulas added after `VP SUSPEND COMPUTING` command is executed are not calculated.

In *vpAreaName*, pass the name of the 4D View Pro area. If you pass a name that does not exist, an error is returned.

The 4D View Pro calculation service maintains a counter of suspend/resume actions. Therefore, each execution of `VP SUSPEND COMPUTING` command must be balanced by a corresponding execution of the `VP RESUME COMPUTING` command. Any formula impacted by modifications made while calculations are suspended will be recalculated when the command is executed.

### Example

You've added two buttons to the form so that the user can suspend/resume calculations:

The Suspend Computing button code:

```
//pause calculations while users enter information
If (FORM Event.code=On Clicked)

    VP SUSPEND COMPUTING ("ViewProArea")

End if
```

```
If (FORM Event.code=On Clicked)
    VP RESUME COMPUTING ("ViewProArea")
End if
```

## See also

[VP RECOMUTE FORMULAS](#)

[VP RESUME COMPUTING](#)

## Classes

The following classes can be used in 4D View Pro.

### LineBorder

#### .color

**.color** : Text

The `.color` property is the [color](#) of the border. Default = black.

#### .style

**.style** : Integer

The `.style` property is the [style](#) of the border. Default = empty.

### TableColumn

#### .dataField

**.dataField** : Text

The `.dataField` property contains the table column's property name in the data context.

#### .formatter

**.formatter** : Text

The `.formatter` property contains the table column's formatter.

#### .name

**.name** : Text

The `.name` property contains the table column's name (mandatory).

### TableOptions

#### .allowAutoExpand

**.allowAutoExpand** : Boolean

The `.allowAutoExpand` property indicates whether to expand columns or rows of the table when values are added in empty adjacent cells. Default = True

#### .bandColumns

**.bandColumns** : Boolean

The `.bandColumns` property indicates whether to display an alternating column style. Default = False

#### .bandRows

## **.bandRows** : Boolean

The `.bandRows` property indicates whether to display an alternating row style. Default = True

## **.highlightLastColumn**

### **.highlightLastColumn** : Boolean

The `.highlightLastColumn` property indicates whether to highlight the last column. Default = False

## **.highlightFirstColumn**

### **.highlightFirstColumn** : Boolean

The `.highlightFirstColumn` property indicates whether to highlight the first column. Default = False

## **.showFooter**

### **.showFooter** : Boolean

The `.showFooter` property indicates whether to display a footer. Default = False

## **.showHeader**

### **.showHeader** : Boolean

The `.showHeader` property indicates whether to display a header. Default = True

## **.showResizeHandle**

### **.showResizeHandle** : Boolean

The `.showResizeHandle` property indicates whether to display the resize handle for tables that don't have a `source`. Default = False

## **.tableColumns**

### **.tableColumns** : Collection

The `.tableColumns` property is a collection of [cs.ViewPro.TableColumn](#) objects used to create the table's columns.

## **.theme**

### **.theme** : [cs.ViewPro.TableThemeOptions](#)

The `.theme` property defines a table theme. Can also be a text (name of a native SpreadJS theme).

See the [native SpreadJS themes](#).

## **.useFooterDropDownList**

### **.useFooterDropDownList** : Boolean

The `.useFooterDropDownList` property indicates whether to use a dropdown list in footer cells that calculate the total value of a column. Default = False

## TableStyle

### .backColor

**.backColor** : Text

The `.backColor` property is the [background color](#) of the table.

### .forecolor

**.forecolor** : Text

The `.forecolor` property is the [foreground color](#) of the table.

### .font

**.font** : Text

The `.font` property is the font name (see [Fonts and text](#)) of the table.

### .textDecoration

**.textDecoration** : Integer

The `.textDecoration` property is the text decoration of the table (see [Fonts and text](#)).

### .borderLeft

**.borderLeft** : [cs.ViewPro.LineBorder](#)

The `.borderLeft` property is the left border line of the table .

### .borderRight

**.borderRight** : [cs.ViewPro.LineBorder](#)

The `.borderRight` property is the right border line of the table .

### .borderBottom

**.borderBottom** : [cs.ViewPro.LineBorder](#)

The `.borderBottom` property is the bottom border line of the table .

### .borderHorizontal

**.borderHorizontal** : [cs.ViewPro.LineBorder](#)

The `.borderHorizontal` property is the horizontal border line of the table .

### .borderVertical

**.borderVertical** : [cs.ViewPro.LineBorder](#)

The `.borderVertical` property is the vertical border line of the table .

## TableTheme

### .bandRows

## **.bandRows** : Boolean

The `.bandRows` property indicates whether to display an alternating row style.

## **.bandColumns**

### **.bandColumns** : Boolean

The `.bandColumns` property indicates whether to display an alternating column style.

## **.highlightLastColumn**

### **.highlightLastColumn** : Boolean

The `.highlightLastColumn` property indicates whether to highlight the last column.

## **.highlightFirstColumn**

### **.highlightFirstColumn** : Boolean

The `.highlightFirstColumn` property indicates whether to highlight the first column.

## **.theme**

### **.theme** : [cs.ViewPro.TableThemeOptions](#)

### **.theme** : Text

The `.theme` property defines a table theme. If Text: name of a [native SpreadJS theme](#).

## **TableThemeOptions**

### **.firstColumnStripSize**

#### **.firstColumnStripSize** : Integer

The `.firstColumnStripSize` property is the size of the first alternating column.  
Default=1

### **.firstColumnStripStyle**

#### **.firstColumnStripStyle** : [cs.ViewPro.TableStyle](#)

The `.firstColumnStripStyle` property is the style of the first alternating column.

### **.firstFooterCellStyle**

#### **.firstFooterCellStyle** : [cs.ViewPro.TableStyle](#)

The `.firstFooterCellStyle` property is the style of the first footer cell.  
"highlightFirstColumn" must be true.

### **.firstHeaderCellStyle**

#### **.firstHeaderCellStyle** : [cs.ViewPro.TableStyle](#)

The `.firstHeaderCellStyle` property is the style of the first header cell.  
"highlightFirstColumn" must be true.

### **.firstRowStripSize**

## **.firstRowStripSize : Integer**

The `.firstRowStripSize` property is the size of the first alternating column.  
Default=1.

## **.firstRowStripStyle**

### **.firstRowStripStyle : [cs.ViewPro.TableStyle](#)**

The `.firstRowStripStyle` property is the first alternating row style.

## **.footerRowStyle**

### **.footerRowStyle : [cs.ViewPro.TableStyle](#)**

The `.footerRowStyle` property is the default style of the footer area.

## **.headerRowStyle**

### **.headerRowStyle : [cs.ViewPro.TableStyle](#)**

The `.headerRowStyle` property is the default style of the header area.

## **.highlightFirstColumnStyle**

### **.highlightFirstColumnStyle : [cs.ViewPro.TableStyle](#)**

The `.highlightFirstColumnStyle` property is the style of the first column.  
"highlightFirstColumn" must be true.

## **.highlightLastColumnStyle**

### **.highlightLastColumnStyle : [cs.ViewPro.TableStyle](#)**

The `.highlightLastColumnStyle` property is the style of the last column.  
"highlightLastColumn" must be true.

## **.lastFooterCellStyle**

### **.lastFooterCellStyle : [cs.ViewPro.TableStyle](#)**

The `.lastFooterCellStyle` property is the style of the last footer cell.  
"highlightLastColumn" must be true.

## **.lastHeaderCellStyle**

### **.lastHeaderCellStyle : [cs.ViewPro.TableStyle](#)**

The `.lastHeaderCellStyle` property is the style of the last header cell.  
"highlightLastColumn" must be true.

## **.name**

### **.name : Text**

The `.name` property is the name of a [native SpreadJS theme](#).

## **.secondColumnStripSize**

### **.secondColumnStripSize : Integer**

The `.secondColumnStripSize` property is the size of the second alternating column.  
Default=1

### **.secondColumnStripStyle**

**.secondColumnStripStyle** : [cs.ViewPro.TableStyle](#)

The `.secondColumnStripStyle` property is the style of the second alternating column.

### **.secondRowStripSize**

**.secondRowStripSize** : Integer

The `.secondRowStripSize` property is the size of the second alternating column.  
Default=1.

### **.secondRowStripStyle**

**.secondRowStripStyle** : [cs.ViewPro.TableStyle](#)

The `.secondRowStripStyle` property is the second alternating row style.

### **.wholeTableStyle**

**.wholeTableStyle** : [cs.ViewPro.TableStyle](#)

The `.wholeTableStyle` property is the default style of the data area.

# Advanced programming with Javascript

A 4D View Pro Area is a [Web Area form object](#) that uses the [embedded web rendering engine](#). As such, it behaves just like any other web area, and you can get it to execute Javascript code by calling the [WA\\_Evaluate\\_Javascript](#) 4D command.

Since 4D View Pro is powered by the [SpreadJS spreadsheet solution](#), you can also call SpreadJS Javascript methods in 4D View Pro areas.

## Hands-on example: Hiding the Ribbon

Since 4D View Pro is a web area, you can select a webpage element and modify its behavior using Javascript. The following example hides the spreadJS [Ribbon](#):

```
//Button's object method  
  
var $js; $answer : Text  
  
$js:="document.getElementsByClassName('ribbon')  
[0].setAttribute('style','display: none');"  
  
$js+="window.dispatchEvent(new Event('resize'));"  
  
$answer:=WA Evaluate JavaScript(*; "ViewProArea"; $js)
```

## Calling SpreadJS Javascript methods

You can tap into the SpreadJS library of Javascript methods and call them directly to control your spreadsheets.

4D has a built-in `Utils.spread` property that gives access to the spreadsheet document (also called workbook) inside the 4D View Pro area, making it simpler to call the SpreadJS [Workbook methods](#).

### Example

The following code undoes the last action in the spreadsheet:

```
WA Evaluate JavaScript(*; "ViewProArea";  
"Utils.spread.undoManager().undo()")
```

## 4D View Pro Tips repository

[4D-View-Pro-Tips](#) is a GitHub repository that contains a project full of useful functions, allowing to manage floating pictures, sort columns or rows, create a custom culture, and much more! Feel free to clone it and experiment with the project.

# Developing Components

A 4D component is a set of 4D functions, methods, and forms representing one or more functionalities that can be [installed and used in 4D applications](#). For example, you can develop a 4D e-mail component that manages every aspect of sending, receiving and storing e-mails in 4D applications.

You can develop 4D components for your own needs and keep them private. You can also [share your components with the 4D community](#).

## Definitions

- **Matrix Project:** 4D project used for developing the component. The matrix project is a standard project with no specific attributes. A matrix project forms a single component.
- **Host Project:** Application project in which a component is installed and used.
- **Component:** Matrix project that can be compiled or [built](#), copied into the [Components](#) folder of the host application and whose contents are used in the host application.

## Basics

Creating and installing 4D components is carried out directly from 4D:

- To install a component, you simply need to copy the component files into the [Components folder of the project](#). You can use aliases or shortcuts.
- A project can be both a matrix and a host, in other words, a matrix project can itself use one or more components. However, a component cannot use "sub-components" itself.
- A component can call on most of the 4D elements: classes, functions, project methods, project forms, menu bars, choice lists, and so on. It cannot call database methods and triggers.
- You cannot use the datastore, standard tables, or data files in 4D components. However, a component can create and/or use tables, fields and data files using mechanisms of external databases. These are separate 4D databases that you work with using SQL commands.
- A host project running in interpreted mode can use either interpreted or compiled components. A host project running in compiled mode cannot use interpreted components. In this case, only compiled components can be used.

## Scope of language commands

Except for [Unusable commands](#), a component can use any command of the 4D language.

When commands are called from a component, they are executed in the context of the component, except for the `EXECUTE METHOD` or `EXECUTE FORMULA` command that use the context of the method specified by the command. Also note that the read commands of the "Users and Groups" theme can be used from a component but will read the users and groups of the host project (a component does not have its own users and groups).

The `SET DATABASE PARAMETER` and `Get database parameter` commands are an exception: their scope is global to the application. When these commands are called from a component, they are applied to the host application project.

Furthermore, specific measures have been specified for the `Structure file` and `Get 4D folder` commands when they are used in the framework of components.

The `COMPONENT LIST` command can be used to obtain the list of components that are loaded by the host project.

## Unusable commands

The following commands are not compatible for use within a component because they modify the structure file — which is open in read-only. Their execution in a component will generate the error -10511, "The CommandName command cannot be called from a component":

- `ON EVENT CALL`
- `Method called on event`
- `SET PICTURE TO LIBRARY`
- `REMOVE PICTURE FROM LIBRARY`
- `SAVE LIST`
- `ARRAY TO LIST`
- `EDIT FORM`
- `CREATE USER FORM`
- `DELETE USER FORM`
- `CHANGE PASSWORD`
- `EDIT ACCESS`
- `Set group properties`
- `Set user properties`
- `DELETE USER`
- `CHANGE LICENSES`
- `BLOB TO USERS`
- `SET PLUGIN ACCESS`

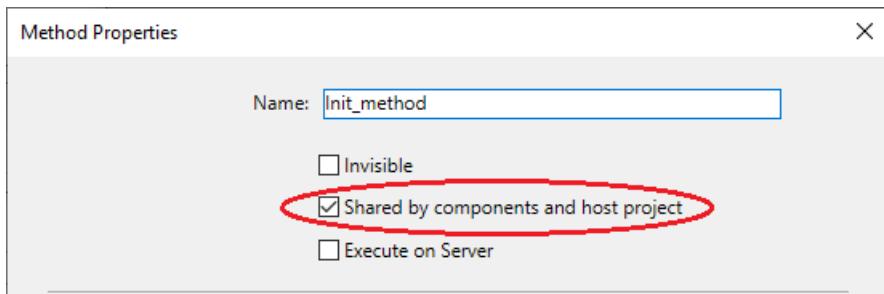
### Notes:

- The `Current form table` command returns `Nil` when it is called in the context of a project form. Consequently, it cannot be used in a component.
- SQL data definition language commands (`CREATE TABLE`, `DROP TABLE`, etc.) cannot be used on the component project. However, they are supported with external databases (see `CREATE DATABASE` SQL command).

## Sharing of project methods

All the project methods of a matrix project are by definition included in the component (the project is the component), which means that they can be called and executed within the component.

On the other hand, by default these project methods will not be visible, and they can't be called in the host project. In the matrix project, you must explicitly designate the methods that you want to share with the host project by checking the **Shared by components and host project** box in the method properties dialog box:



Shared project methods can be called in the code of the host project (but they cannot be modified in the Code Editor of the host project). These methods are **entry points** of the component.

Conversely, for security reasons, by default a component cannot execute project methods belonging to the host project. In certain cases, you may need to allow a component to access the project methods of your host project. To do this, you must explicitly designate which project methods of the host project you want to make accessible to the components (in the method properties, check the **Shared by components and host project** box).

Once the project methods of the host projects are available to the components, you can execute a host method from inside a component using the `EXECUTE FORMULA` or `EXECUTE METHOD` commands. For example:

```
// Host Method
component_method("host_method_name")
// component_method
C_TEXT($1)
EXECUTE METHOD($1)
```

An interpreted host database that contains interpreted components can be compiled or syntax checked if it does not call methods of the interpreted component. Otherwise, a warning dialog box appears when you attempt to launch the compilation or a syntax check and it will not be possible to carry out the operation.

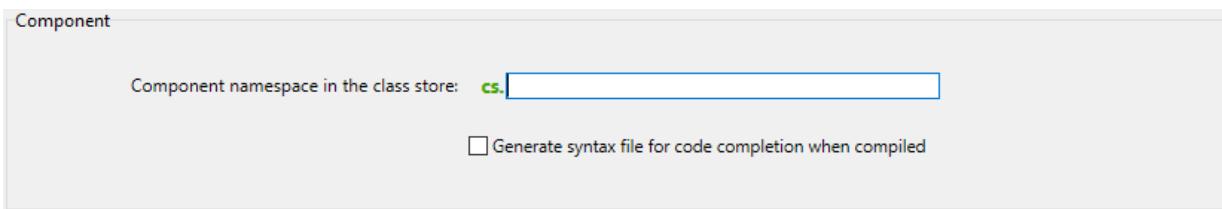
Keep in mind that an interpreted method can call a compiled method, but not the reverse, except via the use of the `EXECUTE METHOD` and `EXECUTE FORMULA` commands.

## Sharing of classes and functions

By default, component classes and functions cannot be called from the 4D Code Editor of the host project. If you want your component classes and functions to be exposed in the host projects, you need to declare a component namespace. Additionally, you can control how component classes and functions are suggested in the host Code Editor.

### Declaring the component namespace

To allow classes and functions of your component to be exposed in the host projects, enter a value in the [Component namespace in the class store option in the General page](#) of the matrix project Settings. By default, the area is empty: component classes are not available outside of the component context.



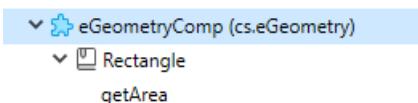
A *namespace* ensures that no conflict emerges when a host project uses different components that have classes or functions with identical names. A component namespace must be compliant with [property naming rules](#).

When you enter a value, you declare that component classes and functions will be available in the [user class store \(cs\)](#) of the host project's code, through the `cs.` `<value>` namespace. For example, if you enter "eGeometry" as component namespace, assuming that you have created a `Rectangle` class containing a `getArea()` function, once your project is installed as a component, the developer of the host project can write:

```
//in host project
var $rect: cs.eGeometry.Rectangle
$rect:=cs.eGeometry.Rectangle.new(10;20)
$area:=$rect.getArea()
```



The namespace of a [compiled](#) component will be added between parentheses after the component name in the [Component Methods page](#) of the host projects:



Of course, it is recommended to use a distinguished name to avoid any conflict. If a user class with the same name as a component already exists in the project, the user class is taken into account and the component classes are ignored.

A component's ORDA classes are not available in its host project. For example, if there is a dataclass called Employees in your component, you will not be able to use a "cs.Mycomponent.Employee" class in the host project.

## Hidden classes

Just like in any project, you can create hidden classes and functions in the component by prefixing names with an underscore ("\_"). When a [component namespace is defined](#), hidden classes and functions of the component will not appear as suggestions when using code completion.

Note however that they can still be used if you know their names. For example, the following syntax is valid even if the `Rectangle` class is hidden:

```
$rect:=cs.eGeometry._Rectangle.new(10;20)
```

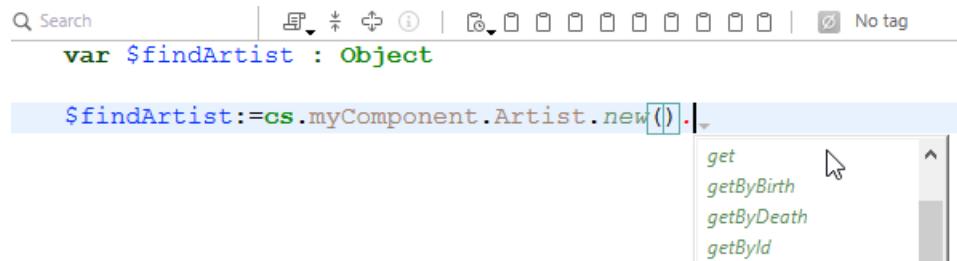
Non-hidden functions inside a hidden class appear as suggestions when you use code completion with a class that [inherits](#) from it. For example, if a component has a `Teacher` class that inherits from a `Person` class, code completion for `Teacher` suggests non-hidden functions from `Person`.

## Code completion for compiled components

To make your component easier to use for developers, you can check the [Generate](#)

[syntax file for code completion when compiled option in the General page](#) of the matrix project Settings.

A syntax file (JSON format) is then automatically created during the compilation phase, filled with the syntax of your component's classes, functions, and [exposed methods](#), and placed in the `\Resources\en.lproj` folder of the component project. 4D uses the contents of that syntax file to generate contextual help in the code editor, such as code completion and function syntax:



If you don't enter a [component namespace](#), the resources for the classes and exposed methods are not generated even if the syntax file option is checked.

## Passing variables

The local, process and interprocess variables are not shared between components and host projects. The only way to modify component variables from the host project and vice versa is using pointers.

Example using an array:

```
//In the host project:  
ARRAY INTEGER (MyArray;10)  
AMethod (->MyArray)  
  
//In the component, the AMethod project method contains:  
APPEND TO ARRAY ($1->;2)
```

Examples using variables:

```
C_TEXT (myvariable)  
component_method1 (->myvariable)  
C_POINTER ($p)  
$p:=component_method2 (...)
```

Without a pointer, a component can still access the value of a host database variable (but not the variable itself) and vice versa:

```
//In the host database  
C_TEXT ($input_t)  
$input_t:="DoSomething"  
component_method ($input_t)  
// component_method gets "DoSomething" in $1 (but not the $input_t  
variable)
```

When you use pointers to allow components and the host project to communicate, you need to take the following specificities into account:

- The `Get pointer` command will not return a pointer to a variable of the host project if it is called from a component and vice versa.
- The component architecture allows the coexistence, within the same interpreted project, of both interpreted and compiled components (conversely, only compiled

components can be used in a compiled project). In order to use pointers in this case, you must respect the following principle: the interpreter can unpoint a pointer built in compiled mode; however, in compiled mode, you cannot unpoint a pointer built in interpreted mode. Let's illustrate this principle with the following example: given two components, C (compiled) and I (interpreted), installed in the same host project.

- If component C defines the `myCvar` variable, component I can access the value of this variable by using the pointer `->myCvar`.
- If component I defines the `myIvar` variable, component C cannot access this variable by using the pointer `->myIvar`. This syntax causes an execution error.
- The comparison of pointers using the `RESOLVE POINTER` command is not recommended with components since the principle of partitioning variables allows the coexistence of variables having the same name but with radically different contents in a component and the host project (or another component). The type of the variable can even be different in both contexts. If the `myptr1` and `myptr2` pointers each point to a variable, the following comparison will produce an incorrect result:

```
RESOLVE POINTER (myptr1;vVarName1;vtablenum1;vfieldnum1)
RESOLVE POINTER (myptr2;vVarName2;vtablenum2;vfieldnum2)
If (vVarName1=vVarName2)
  //This test returns True even though the variables are different
```

In this case, it is necessary to use the comparison of pointers:

```
If (myptr1==myptr2) //This test returns False
```

## Error handling

An [error-handling method](#) installed by the `ON ERR CALL` command only applies to the running application. In the case of an error generated by a component, the `ON ERR CALL` error-handling method of the host project is not called, and vice versa.

## Access to tables of the host project

Although components cannot use tables, pointers can allow host projects and components to communicate with each other. For example, here is a method that could be called from a component:

```
// calling a component method
methCreateRec (->[PEOPLE];->[PEOPLE]Name;"Julie Andrews")
```

Within the component, the code of the `methCreateRec` method:

```
C_POINTER($1) //Pointer on a table in host project
C_POINTER($2) //Pointer on a field in host project
C_TEXT($3) // Value to insert

$stablepointer:=$1
$fieldpointer:=$2
CREATE RECORD ($stablepointer->)

$fieldpointer->:=$3
SAVE RECORD ($stablepointer->)
```

In the context of a component, 4D assumes that a reference to a table form is a reference to the host table form (as components can't have tables.)

## Use of tables and fields

A component cannot use the tables and fields defined in the 4D structure of the matrix project. However, you can create and use external databases, and then use their tables and fields according to your needs. You can create and manage external databases using SQL. An external database is a 4D project that is independent from the main 4D project, but that you can work with from the main 4D project. Using an external database means temporarily designating this database as the current database, in other words, as the target database for the SQL queries executed by 4D. You create external databases using the SQL `CREATE DATABASE` command.

### Example

The following code is included in a component and performs three basic actions with an external database:

- creates the external database if it does not already exist,
- adds data to the external database,
- reads data from the external database.

Creating the external database:

```
<>MyDatabase:=Get 4D folder+"\MyDB" // (Windows) stores the data in an
authorized directory
Begin SQL
    CREATE DATABASE IF NOT EXISTS DATAFILE :[<>MyDatabase];
    USE DATABASE DATAFILE :[<>MyDatabase];
    CREATE TABLE IF NOT EXISTS KEEPIT
    (
        ID INT32 PRIMARY KEY,
        kind VARCHAR,
        name VARCHAR,
        code TEXT,
        sort_order INT32
    );

    CREATE UNIQUE INDEX id_index ON KEEPIT (ID);

    USE DATABASE SQL_INTERNAL;

End SQL
```

Writing in the external database:

```
$Ptr_1:=$2 // retrieves data from the host project through pointers
$Ptr_2:=$3
$Ptr_3:=$4
$Ptr_4:=$5
$Ptr_5:=$6
Begin SQL

    USE DATABASE DATAFILE :[<>MyDatabase];

    INSERT INTO KEEPIT
    (ID, kind, name, code, sort_order)
    VALUES
    (:[$Ptr_1], :[$Ptr_2], :[$Ptr_3], :[$Ptr_4], :[$Ptr_5]);

    USE DATABASE SQL_INTERNAL;

End SQL
```

Reading from an external database:

```
$Ptr_1:=$2 // accesses data of the host project through pointers  
$Ptr_2:=$3  
$Ptr_3:=$4  
$Ptr_4:=$5  
$Ptr_5:=$6
```

Begin SQL

```
USE DATABASE DATAFILE :[<>MyDatabase];  
  
SELECT ALL ID, kind, name, code, sort_order  
FROM KEEPIT  
INTO :$Ptr_1, :$Ptr_2, :$Ptr_3, :$Ptr_4, :$Ptr_5;  
  
USE DATABASE SQL_INTERNAL;
```

End SQL

## Use of forms

- Only “project forms” (forms that are not associated with any specific table) can be used in a component. Any project forms present in the matrix project can be used by the component.
- A component can call table forms of the host project. Note that in this case it is necessary to use pointers rather than table names between brackets [] to specify the forms in the code of the component.

If a component uses the `ADD RECORD` command, the current Input form of the host project will be displayed, in the context of the host project. Consequently, if the form includes variables, the component will not have access to them.

- You can publish component forms as subforms in the host projects. This means that you can, more particularly, develop components offering graphic objects. For example, Widgets provided by 4D are based on the use of subforms in components.

In the context of a component, any referenced project form must belong to the component. For example, inside a component, referencing a host project form using `DIALOG` or `Open form window` will throw an error.

## Use of resources

Components can use resources located in the Resources folder of the component.

Automatic mechanisms are operational: the XLIFF files found in the Resources folder of a component will be loaded by this component.

In a host project containing one or more components, each component as well as the host projects has its own “resources string.” Resources are partitioned between the different projects: it is not possible to access the resources of component A from component B or the host project.

## Executing initialization code

A component can execute 4D code automatically when opening or closing the host database, for example in order to load and/or save the preferences or user states related to the operation of the host database.

Executing initialization or closing code is done by means of the `On Host Database Event` database method.

For security reasons, you must explicitly authorize the execution of the `On Host Database Event` database method in the host database in order to be able to call it. To do this, you must check the [Execute "On Host Database Event" method of the components option](#) in the Security page of the Settings.

## Protection of components: compilation

By default, all the code of a matrix project installed as a component is potentially visible from the host project. In particular:

- The shared project methods are found on the Methods Page of the Explorer and can be called in the methods of the host project. Their contents can be selected and copied in the preview area of the Explorer. They can also be viewed in the debugger. However, it's not possible to open them in the Code Editor or modify them.
- The other project methods of the matrix project do not appear in the Explorer but they too can be viewed in the debugger of the host project.
- The non-hidden classes and functions can be viewed in the debugger [if a namespace is declared](#).

To protect the code of a component effectively, simply [compile and build](#) the matrix project and provide it in the form of a .4dz file. When a compiled matrix project is installed as a component:

- The shared project methods, classes and functions can be called in the methods of the host project and are also visible on the Methods Page of the Explorer. However, their contents will not appear in the preview area and in the debugger.
- The other project methods of the matrix project will never appear.

## Sharing components

We encourage you to support the 4D developer community by sharing your components, preferably on the [GitHub platform](#). We recommend that you use the `4d-component` topic to be correctly referenced.

# Developing Plug-ins

## Why the need for a plug-in?

Although 4D provides hundred of built-in methods used to manipulate objects, records and implement user interface, some special use or feature (sometimes platform dependant) may be needed: one may need ODBC under Windows, another may need Apple services under macOS, while yet another may want to implement specific statistics tools, social network login, payment platform, file access over the network, a special user interface, or a private picture structure.

It is obvious that covering all areas of both the macOS and Windows operating systems by way of 4D commands would certainly lead to a product with thousands of commands, and at the same time, most users would have no need for such a large set of capabilities. Also, creating such an all-encompassing tool would make the 4D environment incredibly complex and would take most users months of study before useful results could be expected.

The modular nature of the 4D environment allows the creation of basic applications but does not preclude the development of highly complex systems. The 4D Plug-in architecture opens the 4D environment to any type of application or user. 4D Plug-ins multiply that application or user's power and productivity.

## What is a plug-in and what can it do?

A plug-in is a piece of code that 4D launches at start up. It adds functionality to 4D and thus increases its capacity.

Usually, a plug-in does things that:

- 4D cannot do (ie, specific platform technology),
- will be very hard to write just using 4D,
- are only available as Plug-in Entrypoint

A plug-in usually contains a set of routines given to the 4D Developer. It can handle an External Area and run an external process.

- A **plug-in routine** is a routine written in native language (usually C or C++) that causes an action.
- An **external area** is a part of a form that can display almost everything and interact with the user when necessary.
- An **external process** is a process that runs alone, usually in a loop, doing almost everything it wants. All process code belongs to the plug-in, 4D is simply present to receive/send events to the process.

## Important note

A plug-in can be very simple, with just one routine performing a very small task, or it can be very complex, involving hundred of routines and areas. There is virtually no limit to what a plug-in can do, however every plug-in developer should remember that a plug-in is a "sample" piece of code. It is the plug-in that runs within 4D, not the opposite. As a piece of code, it is the host of 4D; it is not a stand-alone application. It shares CPU time and memory with 4D and other plug-ins, thus, it should be a polite code, using just what is necessary to run. For example, in long loops, a plug-in should call `PA_Yield()` to give time to the 4D scheduler unless its task is critical for both it

and the application.

## How to create a plug-in?

4D provides on GitHub an open-source [plug-in SDK](#), containing the 4D Plugin API and the 4D Plugin Wizard:

- the [4D Plugin API](#), written in C, adds more than 400 functions that help you to easily create your own plug-ins to add new functionnalities to your 4D application. 4D Plug-in API functions manage all the interactions between the 4D application and your plug-in.
- The [4D Plugin Wizard](#) is an essential tool that simplifies the task of developing 4D plug-ins. It writes the code 4D needs to correctly load and interact with a plug-in, allowing you to concentrate on your own code.

## Sharing plug-ins

We encourage you to support the 4D developer community by sharing your plug-ins, preferably on the [GitHub platform](#). We recommend that you use the `4d-plugin` topic to be correctly referenced.

# Web Applications

Guides for developing Web applications with 4D

## Web Server

[3 items](#)

## Web Development

[7 items](#)

## REST Server

[3 items](#)

## Web Server

4D in local mode, 4D in remote mode and 4D Server include a web server engine (aka http server) that enables you to design and publish powerful web applications that can make the most of your 4D databases.

## Easy Monitoring

You can start or stop publication of the web application at any time. To do so, you just need to select a menu command or execute a single line of code.

Monitoring the 4D web server is easy and can be done using the 4D Server administration window or through [special URLs](#).

## Ready-to-use

The 4D web server automatically creates a default root folder and a default home page for an instantaneous availability.

## Security

Data security is present at every stage of the 4D web server implementations. Security levels are scalable and default settings usually select the most secure options. The 4D web server security is based upon the following elements:

- Extended support of the [TLS Protocol \(HTTPS\)](#),
- **Authentication:** flexible and customizable [authentication features](#) based upon built-it settings as well as fallback database methods ([On Web Authentication](#) for the web server and [On REST Authentication](#) for the REST server),
- **Control of exposed contents:** only elements that you expose explicitly can be available from direct web or REST requests. You must declare:
  - [Project methods](#) exposed through HTTP requests
  - [ORDA functions](#) exposed through REST requests
  - [Tables and fields](#) that you don't want to be available to REST requests.
- **Sandboxing** through the definition of a [HTML Root](#) folder by default,
- **Control of server resource usage** (e.g. [maximum concurrent web processes](#) option).

For a general overview of 4D's security features, see the [4D Security guide](#).

## User Sessions

The 4D web server includes complete automatic features for easily managing [web sessions](#) (user sessions) based on cookies.

## Gateway to REST Requests

The 4D web server allows accessing data stored in your 4D applications through REST requests. REST requests provide direct access to any database operation such as adding, reading, editing, ordering, or searching data.

REST requests are detailed in the [REST server](#) section.

## Extended settings

The 4D web server configuration is defined through a comprehensive set of application-level settings that can also be customized for the session using the `webServer` object properties or the `WEB SET OPTION` command.

## Templates and URLs

The 4D web server supports access to data stored in your 4D applications through template pages and specific URLs.

- Template pages contain [special tags](#) that initiate web server processing at the time when they are sent to browsers.
- [specific URLs](#) enable 4D to be called in order to execute any action; these URLs can also be used as form actions to trigger processing when the user posts HTML forms.

## Dedicated Database Methods

`On Web Authentication`, `On Web Connection`, as well as `On REST Authentication` database methods are the entry points of requests in the web server; they can be used to evaluate and route any type of request.

# Configuration

The 4D web server settings include security parameters, listening ports, defaults paths, and various options covering all the server features. 4D provides default values for every settings.

## Where to configure settings?

There are different ways to configure the 4D web server settings, depending on the scope and the server you want to set:

Setting location	Scope	Involved web server
<a href="#">webServer object</a>	Temporary (current session)	Any web server, including component web servers
<code>WEB SET OPTION</code> or a <code>WEB XXX</code> command	Temporary (current session)	Main server
<a href="#">Settings dialog box (Web pages)</a>	Permanent (all sessions, stored on disk)	Main server

Some settings are not available from all locations.

## Cache

Can be set with	Name	Comments
Settings dialog box	<a href="#">Configuration page/Use the 4D Web cache</a>	
Settings dialog box	<a href="#">Configuration page/Page Cache Size</a>	

Enables and configures the web page cache.

The 4D web server has a cache that allows you to load static pages, GIF images, JPEG images (<512 kb) and style sheets (.css files) in memory, as they are requested. Using the cache allows you to significantly increase the web server's performance when sending static pages. The cache is shared between all the web processes. When the cache is enabled, the 4D Web server looks for any static page requested by the browser in the cache first. If it finds the page, it sends it immediately. If not, 4D loads the page from disk and places it in the cache.

You can modify the size of the cache in the **Pages Cache Size** area. The value you set depends on the number and size of your website's static pages, as well as the resources that the host machines has at its disposal.

While using your web database, you can check the performance of the cache by using the `WEB GET STATISTICS` command. If, for example, you notice that the cache's rate of use is close to 100%, you may want to consider increasing the size that has been allocated to it. The `[/4DSTATS]` and `[/4DHMLSTATS]` URLs allow you to also obtain information about the cache's state.

## Certificate folder

Can be set with	Name	Comments
webServer object	<code>certificateFolder</code>	Text property but can be a <a href="#">4D.Folder</a> object when used with the <i>settings</i> parameter of the <code>start()</code> function.

Folder where the TLS certificate files for the web server are located.

By default with 4D or 4D Server, these files must be placed next to the [project folder](#).

With 4D in remote mode, these files must be located in the local resources folder of the database on the remote machine (see [4D Client Database Folder](#) paragraph of the `Get 4D folder` command). You must copy these files manually on the remote machine.

TLS certificate files are *key.pem* (document containing the private encryption key) and *cert.pem* (document containing the certificate).

## Character Set

Can be set with	Name	Comments
webServer object	<code>characterSet</code>	MIBEnum integer or Name string
<code>WEB SET OPTION</code>	<code>Web character set</code>	MIBEnum integer or Name string
Settings dialog box	<a href="#">Options (II) page/Standard Set</a>	Pop up menu

Defines the set of characters to be used by the 4D web server. The default value actually depends on the language of the OS.

This setting is also used for generating Quick Reports in HTML format .

## Cipher list

Can be set with	Name	Comments
webServer object	<a href="#">cipherSuite</a>	Text

Cipher list used for the secure protocol; sets the priority of ciphering algorithms implemented by the web server. Can be a sequence of strings separated by colons (for example "ECDHE-RSA-AES128-..."). See the [ciphers page](#) on the OpenSSL site.

The default cipher list used by 4D can be modified for the session using the `SET DATABASE PARAMETER` command, in which case the modification applies to the entire 4D application, including the web server, SQL server, client/server connections, as well as the HTTP client and all the 4D commands that make use of the secure protocol.

## CORS Settings

Can be set with	Name	Comments
webServer object	<a href="#">CORSSettings</a>	Collection of objects (List of allowed hosts and methods for the CORS service)
WEB SET OPTION	Web CORS settings	Collection of objects (List of allowed hosts and methods for the CORS service)
Settings dialog box	<a href="#">Options (II) page/Domain names and HTTP methods allowed</a>	Click on the [+] button to add an allowed domain name and its method(s)

List of allowed hosts and methods for the CORS service.

### Domain names (host property)

Domain name or IP address from where external pages are allowed to send data requests to the Server via CORS. Multiple domain attributes can be added to create a white list. Several syntaxes are supported:

- 192.168.5.17:8081
- 192.168.5.17
- 192.168.\*
- 192.168.\*:8081
- <http://192.168.5.17:8081>
- http://\*.myDomain.com
- <http://myProject.myDomain.com>
- \*.myDomain.com
- myProject.myDomain.com
- \*

### HTTP methods allowed (methods property)

Accepted HTTP method(s) for the corresponding CORS host. The following HTTP methods are supported:

- GET
- HEAD
- POST
- PUT
- DELETE
- OPTIONS
- TRACE
- PATCH

Separate each method with a ";" (e.g.: "post;get"). If methods is empty, null, or undefined, all methods are enabled.

### See also

[Enable CORS Service](#)

## Debug log

Can be set with	Name	Comments
webServer object	debugLog	number
WEB SET OPTION	Web debug log	number

Status of the HTTP request log file of the web server ([HTTPDebugLog\\_nn.txt](#)), stored

in the "Logs" folder of the application -- nn is the file number). It is useful for debugging issues related to the Web server. It records each request and each response in raw mode. Whole requests, including headers, are logged; optionally, body parts can be logged as well.

<b>Value</b>	<b>Constant</b>	<b>Description</b>
0	wdl disable	Web HTTP debug log is disabled
1	wdl enable without body	Web HTTP debug log is enabled without body parts (body size is provided in this case)
3	wdl enable with response body	Web HTTP debug log is enabled with body part in response only
5	wdl enable with request body	Web HTTP debug log is enabled with body part in request only
7	wdl enable with all body parts	Web HTTP debug log is enabled with body parts in response and request

## Default Home page

<b>Can be set with</b>	<b>Name</b>	<b>Comments</b>
webServer object	<a href="#">defaultHomepage</a>	Text
WEB SET HOME PAGE		Can be different for each web process
Settings dialog box	<a href="#">Configuration page/Default Home Page</a>	

Designate a default home page for the web server. This page can be static or [semi-dynamic].

By default, when the web server is launched for the first time, 4D creates a home page named "index.html" and puts it in the HTML root folder. If you do not modify this configuration, any browser connecting to the web server will obtain the following page:

You can designate another default home page by entering its pathname.

- The path is relative to the [default HTML root folder](#).
- The path is expressed with the POSIX syntax (folders are separated by a slash ("/"))
- The path must neither start nor end with a slash.

For example, if you want the default home page to be "MyHome.htm", and it is located in the "Web" folder (itself located in the default HTML root folder), use "Web/MyHome.htm".

If you do not specify any default home page, the [On Web Connection](#) database method is called. It is up to you to process the request procedurally.

## Enable CORS Service

<b>Can be set with</b>	<b>Name</b>	<b>Comments</b>
webServer object	<a href="#">CORSEnabled</a>	Boolean, true to enable the CORS (false by default)
WEB SET OPTION	Web CORS enabled	0 (disabled, default) or 1 (enabled)
Settings dialog box	<a href="#">Options (II) page/Enable CORS</a>	Unchecked by default

The 4D web server implements cross-origin resource sharing (CORS) to allow specific Web pages served from another domain to access the current Web application's resources via XHR calls, e.g., using REST. For security reasons, "cross-domain" requests are forbidden at the browser level by default. When enabled, XHR calls (e.g. REST requests) from Web pages outside the domain can be allowed in your application (you need to define the list of allowed addresses in the CORS domain list, see CORS Settings below). In this case, if a non-allowed domain or method sends a cross site request, it is rejected with a "403 - forbidden" error response.

When disabled (default), all cross site requests sent with CORS are ignored.

For more information about CORS, please refer to the [Cross-origin resource sharing page](#) on Wikipedia.

## See also

[CORS Settings](#)

## Enable HTTP

<b>Can be set with</b>	<b>Name</b>	<b>Comments</b>
webServer object	<a href="#">HTTPEnabled</a>	boolean
WEB SET OPTION	Web HTTP enabled	
Settings dialog box	<a href="#">Configuration page/Enable HTTP</a>	

Indicates whether or not the web server will accept non-secure connections.

## Enable HTTPS

<b>Can be set with</b>	<b>Name</b>	<b>Comments</b>
webServer object	<a href="#">HTTPSEnabled</a>	boolean
WEB SET OPTION	Web HTTPS enabled	
Settings dialog box	Configuration page/Enable HTTPS	

Status for communication over HTTPS. This option is described in [this section](#).

## Enable HSTS

<b>Can be set with</b>	<b>Name</b>	<b>Comments</b>
webServer object	<a href="#">HSTSEnabled</a>	Boolean, true to enable HSTS (default is false)
WEB SET OPTION	Web HSTS enabled	0 (disabled, default) or 1 (enabled)

HTTP Strict Transport Security (HSTS) status.

When [HTTPS is enabled](#), keep in mind that if [HTTP is also enabled](#), the browser can

still switch between HTTPS and HTTP (for example, in the browser URL area, the user can replace "https" by "http"). To forbid http redirections, you can [disable HTTP](#), however in this case an error message is displayed to client HTTP requests.

HSTS allows the 4D web server to declare that browsers should only interact with it via secure HTTPS connections. Once activated, the 4D web server will automatically add HSTS-related information to all response headers. Browsers will record the HSTS information the first time they receive a response from the 4D web server, then any future HTTP requests will automatically be transformed into HTTPS requests. The length of time this information is stored by the browser is specified with the Web **HSTS max age** setting.

HSTS requires that [HTTPS is enabled](#) on the server. [HTTP](#) must also be enabled to allow client initial connections.

You can get the current connection mode using the `WEB Is secured connection` command.

## HSTS Max Age

Can be set with	Name	Comments
webServer object	<a href="#">HSTSMaxAge</a>	number in seconds
<code>WEB SET OPTION Web HSTS max age</code>		number in seconds

Specifies the maximum length of time (in seconds) that HSTS is active for each new client connection. This information is stored on the client side for the specified duration. Default value is 63072000 (2 years)

**Warning:** Once HSTS is enabled, client connections will continue to use this mechanism for the specified duration. When you are testing your applications, it is recommended to set a short duration to be able to switch between secured and non-secured connection modes if necessary.

## HTTP Compression Level

Can be set with	Name	Comments
webServer object	<a href="#">HTTPCompressionLevel</a>	
<code>WEB SET OPTION Web HTTP compression level</code>		Applies to Web and Web Service

Compression level for all compressed HTTP exchanges for the 4D web server (client requests or server replies). This setting lets you optimize exchanges by either privileging speed of execution (less compression) or the amount of compression (less speed). The choice of a value depends on the size and type of data exchanged.

Pass 1 to 9 as value where 1 is the fastest compression and 9 the highest. You can also pass -1 to get a compromise between speed and rate of compression. By default, the compression level is 1 (faster compression).

## HTTP Compression Threshold

Can be set with	Name	Comments
webServer object	<a href="#">HTTPCompressionThreshold</a>	
<code>WEB SET OPTION Web HTTP compression threshold</code>		

In the framework of optimized HTTP exchanges, size threshold for requests below which exchanges should not be compressed. This setting is useful in order to avoid

losing machine time by compressing small exchanges.

Pass the size expressed in bytes as value. By default, the compression threshold is set to 1024 bytes.

## HTTP Port

Can be set with	Name	Comments
webServer object	<a href="#">HTTPPort</a>	number
WEB SET OPTION	Web port ID	
Settings dialog box	<a href="#">Configuration page/HTTP Port</a>	

Listening IP (TCP) port number for HTTP. By default, 4D publishes a web application on the regular Web HTTP Port (TCP port), which is port 80. If that port is already used by another web service, you need to change the HTTP Port used by 4D for this database.

In macOS, modifying the HTTP port allows you to start the 4D web server without being the root user of the machine (see [macOS HelperTool](#)).

From a web browser, you need to include the non-default HTTP port number into the address you enter for connecting to the web application. The address must have a suffix consisting of a colon followed by the port number. For example, if you are using the HTTP port number 8080, you will specify "123.4.567.89:8080".

**Warning:** If you use TCP port numbers other than the default numbers (80 for standard HTTP and 443 for HTTPS), be careful not to use port numbers that are defaults for other services that you might want to use simultaneously. For example, if you also plan to use the FTP protocol on your web server machine, do not use the TCP port 20 and 21, which are the default ports for that protocol. Ports numbers below 256 are reserved for well known services and ports numbers from 256 to 1024 are reserved for specific services originated on the UNIX platforms. For maximum security, specify a port number beyond these intervals (for example, in the 2000's or 3000's).

If you specify 0, 4D will use the default HTTP port number 80.

## HTTP Trace

Can be set with	Name	Comments
webServer object	<a href="#">HTTPTrace</a>	Boolean, default = false
WEB SET OPTION	Web HTTP TRACE	Integer, default = 0 (disabled)

HTTP TRACE method activation in the 4D web server. For security reasons, by default the 4D web server rejects HTTP TRACE requests with an error 405. If necessary, you can enable the HTTP TRACE method, in which case the 4D Web server replies to HTTP TRACE requests with the request line, header, and body.

## HTTPS Port

Can be set with	Name	Comments
webServer object	<a href="#">HTTPSPort</a>	number
WEB SET OPTION	Web HTTPS port ID	

## |Settings dialog box|[Configuration page/HTTPS Port](#)||

Listening IP port number for HTTPS connections via TLS. By default, the value is 443 (standard value). See also [HTTP Port](#) for information on port numbers.

## Inactive Process Timeout

Can be set with	Name	Comments
webServer object	<a href="#">inactiveProcessTimeout</a>	
WEB SET OPTION	Web inactive process timeout	
Settings dialog box	<a href="#">Options (I) page/Inactive Process Timeout</a>	Slider

Life duration (in minutes) of inactive processes associated with sessions. At the end of the timeout, the process is killed on the server, the `On Web Close Process` database method is called, then the session context is destroyed.

Default: 480 minutes (pass 0 to restore the default value)

## Inactive Session Timeout

Can be set with	Name	Comments
webServer object	<a href="#">inactiveSessionTimeout</a>	
WEB SET OPTION	Web inactive session timeout	

Life duration (in minutes) of inactive sessions (duration set in cookie). At the end of this period, the session cookie expires and is no longer sent by the HTTP client.

Default: 480 minutes (pass 0 to restore the default value)

## IP Address to listen

Can be set with	Name	Comments
webServer object	<a href="#">IPAddressToListen</a>	
WEB SET OPTION	Web IP address to listen	
Settings dialog box	<a href="#">Configuration page/IP Address</a>	Pop up menu

IP address strings on which the 4D web server will receive HTTP requests (4D local and 4D Server).

By default, no specific address is defined (**Any** value in the Settings dialog box), which means that the server responds to all IP addresses. When you designate a specific address, the server only responds to requests sent to this address. This feature is designed for 4D web servers located on machines with multiple TCP/IP addresses. It is, for example, frequently the case of most host providers.

Possible values: IP address string. Both IPv6 string formats (e.g. "2001:0db8:0000:0000:ff00:0042:8329") and IPv4 string formats (e.g. "123.45.67.89") are supported.

## About IPv6 support

- **No warning when TCP port is occupied**

When the server is set to respond on "Any" IP addresses, if the TCP port is being used by another application, this is not indicated when the server is started. In fact, 4D server does not detect any error in this case because the port remains

free on the IPv6 address. However, it is not possible to access it using the IPv4 address of the machine, nor by means of the local address: 127.0.0.1.

If your 4D server does not seem to be responding on the port defined, you can test the address [::1] on the server machine (equivalent to 127.0.0.1 for IPv6, add [:portNum] to test another port number). If 4D responds, it is likely that another application is using the port in IPv4.

- **IPv4-mapped IPv6 addresses**

To standardize processing, 4D provides a standard hybrid representation of IPv4 addresses in IPv6. These addresses are written with a 96-bit prefix in IPv6 format, followed by 32 bits written in the dot-decimal notation of IPv4. For example, ::ffff:192.168.2.34 represents the IPv4 address 192.168.2.34.

- **Indication of port numbers**

Since IPv6 notation uses colons (:), adding port numbers may lead to some confusion, for example:

```
2001:0DB8::85a3:0:ac1f:8001 // IPv6 address  
2001:0DB8::85a3:0:ac1f:8001:8081 // IPv6 address with port 8081
```

To avoid this confusion, we recommend using the [ ] notation whenever you combine an IPv6 address with a port number, for instance:

```
[2001:0DB8::85a3:0:ac1f:8001]:8081 //IPv6 address with port 8081
```

## Keep Session

Can be set with	Name	Comments
webServer object	<a href="#">keepSession</a>	
WEB SET OPTION	Web keep session	
Settings dialog box	<a href="#">Options (I) page/Legacy sessions (single process sessions)</a>	only in converted projects

Legacy session management enabling status for the 4D web server (deprecated).

When this option is checked, the "Reuse Temporary Contexts" option is automatically checked (and locked).

## Log Recording

Can be set with	Name	Comments
webServer object	<a href="#">logRecording</a>	
WEB SET OPTION	Web log recording	
Settings dialog box	<a href="#">Log (type) page</a>	Pop up menu

Starts or stops the recording of requests received by the 4D web server in the *logweb.txt* file and sets its format. By default, requests are not recorded (0/No Log File). When enabled, the *logweb.txt* file is automatically placed in the Logs folder.

This setting allows you to select the format of this file. Available values are:

Value	Format name	Description
0	No Log File	Default
1	Record in CLF format	Common Log Format - Each line of the file represents a request, such as: host rfc931 user [DD/MMM/YYYY:HH:MM:SS] "request" state length - Each field is separated by a space and each line ends by the CR/LF sequence.
2	Record in DLF format	Combined Log Format - Similar to CLF format but adds two additional HTTP fields at the end of each request: Referer and User-agent.
3	Record in ELF format	Extended Log Format - To be customized in the Settings dialog box
4	Record in WLF format	WebStar Log Format - To be customized in the Settings dialog box

Formats 3 and 4 are custom formats whose contents must be set beforehand in the [Settings dialog box](#). If you use one of these formats without any of its fields having been selected on this page, the log file will not be generated.

## Maximum Concurrent Web Processes

Can be set with	Name	Comments
webServer object	<a href="#">maxConcurrentProcesses</a>	
WEB SET OPTION	Web max concurrent processes	
Settings dialog box	<a href="#">Options (I) page/Maximum Concurrent Web Processes</a>	

Strictly high limit of concurrent web processes that can be simultaneously open on the server when **no sessions** or **legacy sessions** are used (**scalable sessions** support an [unlimited number](#) of preemptive processes). This parameter allows prevention of server saturation as the result of massive number of requests. When the maximum number of concurrent Web processes (minus one) is reached, 4D no longer creates new processes and sends the HTTP status [503 – Service Unavailable](#) to all new requests.

By default, the value is 100. You can set the number anywhere between 10 and 32000.

## Maximum Request Size

Can be set with	Name	Comments
webServer object	<a href="#">maxRequestSize</a>	
WEB SET OPTION	Web maximum requests size	

Maximum size (in bytes) of incoming HTTP requests (POST) that the web server is authorized to process. By default, the value is 2 000 000, i.e. a little less than 2 MB. Passing the maximum value (2 147 483 648) means that, in practice, no limit is set.

This limit is used to avoid web server saturation due to incoming requests that are too large. When a request reaches this limit, the 4D web server rejects it.

Possible values: 500 000 to 2 147 483 648.

## Maximum Session Number

Can be set with	Name	Comments
webServer object	<a href="#">maxSessions</a>	

WEB SET OPTION Web max sessions

Maximum number of simultaneous sessions. When you reach the limit set, the oldest session is closed (and [On Web Close Process](#) database method is called) if the Web server needs to create a new one. The number of simultaneous sessions cannot exceed the [maximum number of Web processes](#) (100 by default).

Default value: 100 (pass 0 to restore the default value).

## Minimum TLS Version

Can be set with	Name	Comments
webServer object	<a href="#">minTLSVersion</a>	number

Minimum TLS version accepted for connections. Connection attempts from clients supporting only versions below the minimum will be rejected.

Possible values:

- 1 = TLSv1\_0
- 2 = TLSv1\_1
- 3 = TLSv1\_2 (default)
- 4 = TLSv1\_3

If modified, the server must be restarted to use the new value.

The minimum TLS version used by 4D can be modified for the session using the [SET DATABASE PARAMETER](#) command, in which case the modification applies to the entire 4D application, including the web server, SQL server and client/server connections.

## Name

Can be set with	Name	Comments
-----------------	------	----------

webServer object [name](#)

Name of the web server application. Useful when component web servers are started.

## OpenSSL Version

Can be set with	Name	Comments
webServer object	<a href="#">openSSLVersion</a>	Read-only

Version of the OpenSSL library used.

## Perfect Forward Secrecy

Can be set with	Name	Comments
webServer object	<a href="#">perfectForwardSecrecy</a>	Boolean, read-only

True if PFS is available on the web server (see [TLS](#) section).

## Reuse temporary contexts (in remote mode)

Can be set with	Name	Comments
Settings dialog box	<a href="#">Options (I) page/Maximum Concurrent Web Processes</a>	

This option is only available when **No sessions** option is checked.

Allows you to optimize the operation of the 4D Web Server in remote mode by reusing web processes created for processing previous web requests. In fact, the web server in 4D needs a specific web process for the handling of each web request; in remote mode, when necessary, this process connects to the 4D Server machine in order to access the data and database engine. It thus generates a temporary context using its own variables, selections, etc. Once the request has been dealt with, this process is killed.

When the **Reuse Temporary Contexts** option is checked, in remote mode 4D maintains the specific web processes and reuses them for subsequent requests. By removing the process creation stage, web server performance is improved.

In return, you must make sure in this case to systematically initialize the variables used in 4D methods in order to avoid getting incorrect results. Similarly, it is necessary to erase any current selections or records defined during the previous request.

This option only has an effect with a 4D web server in remote mode. With a 4D in local mode, all web processes (other than session processes) are killed after their use.

## Robots.txt

Certain robots (query engines, spiders...) scroll through web servers and static pages. If you do not want robots to be able to access your entire site, you can define which URLs they are not allowed to access.

To do so, put the ROBOTS.TXT file at the server's root. This file must be structured in the following manner:

```
User-Agent: <name>
Disallow: <URL> or <beginning of the URL>
```

For example:

```
User-Agent: *
Disallow: /4D
Disallow: /%23%23
Disallow: /GIFS/
```

- “User-Agent: \*” - all robots are affected.
- “Disallow: /4D” - robots are not allowed to access URLs beginning with /4D.
- “Disallow: /%23%23” - robots are not allowed to access URLs beginning with /%23%23.
- “Disallow: /GIFS/” - robots are not allowed to access the /GIFS/ folder or its subfolders.

Another example:

```
User-Agent: *
Disallow: /
```

In this case, robots are not allowed to access the entire site.

## Root Folder

Can be set with	Name	Comments
webServer object	<a href="#">rootFolder</a>	Text property but can be a <a href="#">4D.Folder</a> object when used with the <i>settings</i> parameter of the <code>start()</code> function
WEB SET ROOT FOLDER		

Settings dialog box [Configuration page/Default HTML Root](#)

Path of web server root folder, i.e. the folder in which 4D will search for the static and semi-dynamic HTML pages, pictures, etc., to send to the browsers. The path is formatted in POSIX full path. The web server will need to be restarted in order for the new root folder to be taken into account.

Moreover, the HTML root folder defines, on the web server hard drive, the hierarchical level above which the files will not be accessible. If a requested URL or a 4D command tries to access a file located above the HTML root folder, an error is returned indicating that the file has not been found.

By default, 4D defines a HTML Root folder named **WebFolder**. If it does not already exist, the HTML root folder is physically created on disk at the moment the Web server is launched for the first time. The root folder is created:

- with 4D (local) and 4D Server, at the same level as the [Project folder](#).
- with 4D in remote mode, in the local resources folder.

You can designate another default HTML root folder by entering its pathname.

- The path is relative to the [Project folder](#) (4D local and 4D Server) or to the folder containing the 4D application or software package (4D in remote mode).
- The path is expressed with the POSIX syntax (folders are separated by a slash ("/"))
- To "go up" one level in the folder hierarchy, enter ".." (two periods) before the folder name
- The path must not start with a slash (except if you want the HTML root folder to be the Project or 4D remote folder, but for access to the folders above to be forbidden, in which case you can pass "/" as the root folder).

For example, if you want the HTML root folder to be the "Web" subfolder in the "MyWebApp" folder, enter "MyWebApp/Web".

When the HTML root folder is modified, the cache is cleared so as to not store files whose access is restricted.

## Scalable Sessions

Can be set with	Name	Comments
webServer object	<a href="#">scalableSession</a>	
WEB SET OPTION	Web scalable session	

Settings dialog box [Options \(I\) page/Scalable sessions \(multi-process sessions\)](#)

Scalable session management enabling status for the 4D web server. Web server sessions are detailed in the [User sessions](#) page.

## Session Cookie Domain

Can be set with	Name	Comments
webServer object <a href="#">sessionCookieDomain</a>		WEB SET OPTION Web session cookie domain

Value of the "domain" field of the session cookie. Useful for controlling the scope of the session cookies. If you set, for example, the value "/\*.4d.fr" for this selector, the client will only send a cookie when the request is addressed to the domain ".4d.fr", which excludes servers hosting external static data.

## Session Cookie Name

Can be set with	Name	Comments
webServer object <a href="#">sessionCookieName</a>		WEB SET OPTION Web session cookie name

Name of the cookie used for saving the session ID. Default = "4DSID".

## Session Cookie Path

Can be set with	Name	Comments
webServer object <a href="#">sessionCookiePath</a>		WEB SET OPTION Web session cookie path

"path" field of the session cookie. Used to control the scope of the session cookies. If you set, for example, the value "/4DACTION" for this selector, the client will only send a cookie for dynamic requests beginning with 4DACTION, and not for pictures, static pages, etc.

## Session Cookie SameSite

Can be set with	Name	Comments
webServer object <a href="#">sessionCookieSameSite</a>		

Value of the `SameSite` attribute value of the session cookie. This attribute allows you to declare if your cookie should be restricted to a first-party or same-site context, as a protection from some cross-site request forgery ([CSRF](#)) attacks.

For a detailed description of the `SameSite` attribute, please refer to the [Mozilla documentation](#) or [this web.dev page](#).

Three values are available:

- "Strict" (default `SameSite` attribute value for 4D session cookies): cookies will only be sent in the first-party context, i.e. context matching the domain of the current site, and never to third-party websites.
- "Lax": Cookies are not sent on cross-site subrequests (for example to load images or frames into a third-party site), but are sent when a user is navigating to the origin site (i.e. they follow a link).
- "None": Cookies are sent in all contexts, i.e in responses to both first-party and cross-origin requests. When "None" value is used, the cookie `Secure` attribute must also be set (or the cookie will be blocked).

The `Secure` attribute value of the session cookie is automatically set to "True" if the connection is HTTPS (whatever the `SameSite` attribute value).

It is not recommended to set `SameSite=None` on a HTTP server since the `Secure` attribute will be missing (used in HTTPS only) and cookies will be blocked.

## Use preemptive processes

Can be set with	Name	Comments
Settings dialog box	<a href="#">Options (I) page/Maximum Concurrent Web Processes</a>	

This option enables the preemptive mode for your application's web server code when **No sessions** option is selected (the preemptive mode is always enabled with **scalable sessions**). When this option is checked in this context, the 4D compiler will automatically evaluate the thread-safety property of each piece of [web-related code](#) and return errors in case of incompatibility.

## Deprecated Settings

The following settings are still supported but rely on deprecated features or technologies. It is usually recommended to keep default values.

### Allow database access through 4DSYNC URLs

This option controls the support of HTTP synchronization requests containing deprecated /4DSYNC URLs.

### Session IP Address Validation

This option is not available in [scalable sessions mode](#) (there is no validation).

IP address validation status for session cookies. For security reasons, by default the 4D web server checks the IP address of each request containing a session cookie and rejects it if this address does not match the IP address used to create the cookie. In some specific applications, you may want to disable this validation and accept session cookies, even when their IP addresses do not match. For example when mobile devices switch between Wifi and 4G/5G networks, their IP address will change. In this case, you must pass 0 in this option to allow clients to be able to continue using their Web sessions even when the IP addresses change. Note that this setting lowers the security level of your application. When it is modified, this setting is effective immediately (you do not need to restart the HTTP server).

### Send Extended Characters Directly

When this option is checked, the web server sends extended characters "as is" in semi-dynamic pages, without converting them into HTML entities. This option has shown a speed increase on most foreign operating systems (especially the Japanese system).

### Keep-Alive Connections

The 4D Web Server can use keep-alive connections. The keep-alive option allows you to maintain a single open TCP connection for the set of exchanges between the web browser and the server to save system resources and to optimize transfers.

The **Use Keep-Alive Connections** option enables or disables keep-alive TCP connections for the web server. This option is enabled by default. In most cases, it is advisable to keep this option checked since it accelerates the exchanges. If the web browser does not support connection keep alive, the 4D Web Server automatically switches to HTTP/1.0.

The 4D Web Server keep-alive function concerns all TCP/IP connections (HTTP, HTTPS). Note however that keep-alive connections are not always used for all 4D web processes.

In some cases, other optimized internal functions may be invoked. Keep-alive connections are useful mainly for static pages.

Two options allow you to set how the keep-alive connections work:

- **Number of requests by connection:** Allows you to set the maximum number of requests and responses able to travel over a connection keep alive. Limiting the number of requests per connection allows you to prevent server flooding due to a large number of incoming requests (a technique used by hackers).

The default value (100) can be increased or decreased depending on the resources of the machine hosting the 4D Web Server.

- **Timeout:** This value defines the maximum wait period (in seconds) during which the web server maintains an open TCP connection without receiving any requests from the web browser. Once this period is over, the server closes the connection.

If the web browser sends a request after the connection is closed, a new TCP connection is automatically created. This operation is not visible for the user.

## Administration

4D provides several integrated tools to start, stop, or monitor the integrated web server.

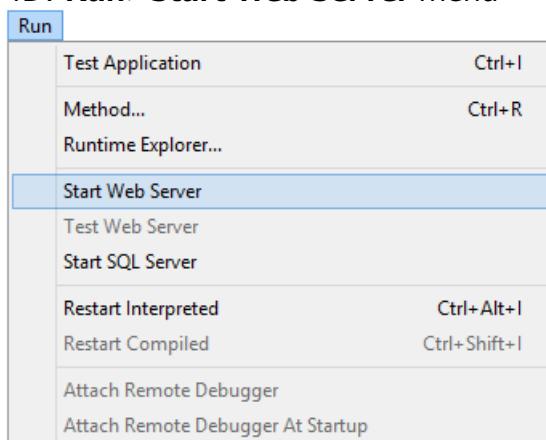
### Starting the 4D Web Server

To be able to launch the web server of 4D or 4D Server, you must have a "4D Web Application" license. For more information, please refer to the [4D Web site](#).

A 4D project can start and monitor a web server for the main (host) application as well as for each hosted component.

The main 4D web server can be started in different ways:

- Using a button/menu command.
  - 4D: **Run>Start Web Server** menu



- 4D Server: **Start HTTP server** button of the HTTP Server page
- Automatically starting it each time the 4D application is opened. To do this, display the **Web¥/Configuration** page of the Settings and select the **Launch Web Server at Startup** check box:
- Programmatically, by calling the `webServer.start()` function or `WEB START SERVER` command.

The web server of any component can be launched by calling the `webServer.start()` function on the component's web server object.

You do not need to relaunch the 4D application to start or stop the web server.

### Stopping the 4D Web Server

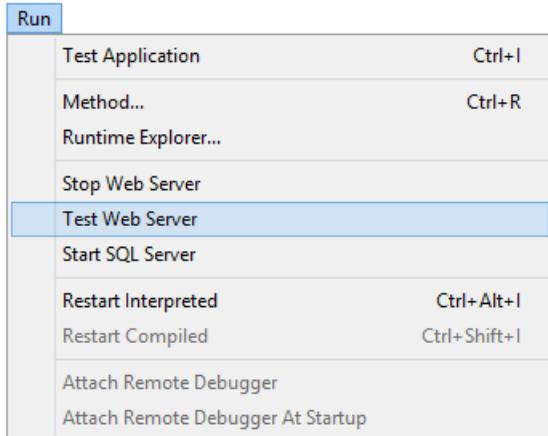
The main 4D web server can be stopped in different ways:

- Using the **Run>Stop Web Server** menu of 4D or the **Stop HTTP server** button of the HTTP Server page of 4D Server (both items show **Start...** when the server is not already started).
- Programmatically, by calling the `webServer.stop()` function or `WEB STOP SERVER` command.

The web server of any component can be stopped by calling the `webServer.stop()` function on the component's web server object.

## Testing the 4D Web Server

The **Test Web Server** command can be used to make sure the built-in web server is functioning correctly (4D only). This command is accessible in the **Run** menu when the web server is launched:



When you select this command, the home page of the website published by the 4D application is displayed in a window of your default web browser:

This command lets you verify that the web server, home page display, etc. work correctly. The page is called using the *localhost* URL, which is the standard shortcut designating the IP address of the machine on which the web browser is executed. The command takes into account the [TCP publication port](#) number specified in the settings.

## Clearing the Cache

At any moment, you can clear the cache of the pages and images that it contains (if, for example, you have modified a static page and you want to reload it in the cache).

To do so, you can:

- 4D: click on the **Clear Cache** button in the [Web/Options \(I\) page](#) of the Settings dialog box.
- 4D Server: click on the **Clear Cache** button in the HTTP page of the 4D Server Administration window.

The cache is then immediately cleared.

You can also use the [/4DCACHECLEAR](#) URL.

## Runtime Explorer

The **Watch** page (**Web** heading) in the Runtime Explorer displays web server information, particularly:

- **Web Cache Usage:** indicates the number of pages present in the web cache as well as its use percentage. This information is only available if the web server is active and if the cache size is greater than 0.
- **Web Server Elapsed Time:** indicates the duration of use (in

hours:minutes:seconds format) of the Web server. This information is only available if the web server is active.

- **Web Hits Count:** indicates the total number of HTTP requests received since the web server boot, as well as an instantaneous number of requests per second (measure taken between two Runtime Explorer updates). This information is only available if the web server is active.

## Administration URLs

Website administration URLs allow you to control the website published on your server. 4D Web Server accepts four particular URLs: */4DSTATS*, */4DHTMLSTATUS*, */4DCACHECLEAR* and */4DWEBTEST*.

*/4DSTATS*, */4DHTMLSTATUS* and */4DCACHECLEAR* are only available to the Designer and Administrator of the database. If the 4D password system has not been activated, these URLs are available to all the users. */4DWEBTEST* is always available.

### **/4DSTATS**

The **/4DSTATS** URL returns several items of information in an HTML table (displayable in a browser):

Item	Description
Cache Current Size	Current size of web server cache (in bytes)
Cache Max Size	Maximum size of cache (in bytes)
Cached Object Max Size	Maximum size of each object in the cache (in bytes)
Cache Use	Percentage of cache used
Cached Objects	Number of objects found in the cache, <b>including pictures</b>

This information can allow you to check the functioning of your server and eventually adapt the corresponding parameters.

The `WEB GET STATISTICS` command allows you to also obtain information about how the cache is being used for static pages.

### **/4DHTMLSTATUS**

The */4DHTMLSTATUS* URL returns, also as an HTML table, the same information as the */4DSTATS* URL. The difference is that the **Cached Objects** field only counts HTML pages (without counting picture files). Moreover, this URL returns the **Filtered Objects** field.

Item	Description
Cache Current Size	Current size of web server cache (in bytes)
Cache Max Size	Maximum size of cache (in bytes)
Cached Object Max Size	Maximum size of each object in the cache (in bytes)
Cache Use	Percentage of cache used
Cached Objects	Number of objects found in the cache, <b>without pictures</b>
Filtered Objects	Number of objects in cache not counted by URL, in particular, pictures

### **/4DCACHECLEAR**

The `/4DCACHECLEAR` URL immediately clears the cache of the static pages and images. It allows you to therefore “force” the update of the pages that have been modified.

## **/4DWEBTEST**

The `/4DWEBTEST` URL is designed to check the web server status. When this URL is called, 4D returns a text file with the following HTTP fields filled:

<b>HTTP Field</b>	<b>Description</b>	<b>Example</b>
Date	current date at the RFC 822 format	Mon, 7 Dec 2020 13:12:50 GMT
Server	4D/version number	4D/18.5.0 (Build 18R5.257368)
User-Agent	name and version @ IP client address	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.183 Safari/537.36 @ 127.0.0.1

## **Logs**

4D allows you to generate two logs of web requests:

- a debug log, useful in the web server development phase (`HTTPDebugLog.txt`),
- a standardized web request log, rather used for statistic purposes (`logweb.txt`).

Both log files are automatically created in the **Logs** folder of the application project.

### **HTTPDebugLog.txt**

The [http debug file](#) can be enabled using the [web server object](#) or the `WEB SET OPTION` command.

This log file records each HTTP request and each response in raw mode. Whole requests, including headers, are logged; optionally, body parts can be logged as well.

The following fields are logged for both Request and Response:

<b>Field name</b>	<b>Description</b>
SocketID	ID of socket used for communication
PeerIP	IPv4 address of host (client)
PeerPort	Port used by host (client)
TimeStamp	Timestamp in milliseconds (since system startup)
ConnectionID	Connection UUID (UUID of VTCPSocket used for communication)
SequenceNumber	Unique and sequential operation number in the logging session

### **logweb.txt**

The [web log recording file](#) can be enabled using the [web server object](#), the `WEB SET OPTION` command, or the **Web/Log (type)** page of the settings. You need to select the log format.

### **CLF/DLF**

Each line of the file represents a request, such as: `host rfc931 user [DD/MMM/YYYY:HH:MM:SS] "request" state length` Each field is separated by a space

and each line ends by the CR/LF sequence (character 13, character 10).

DLF (Combined Log Format) format is similar to CLF (Common Log Format) format and uses exactly the same structure. It simply adds two additional HTTP fields at the end of each request: Referer and User-agent. Here is the description of CLF/DLF formats (not customizable):

<b>Field name</b>	<b>Description</b>
host	IP address of the client (ex. 192.100.100.10)
rfc931	information not generated by 4D, it's always - (a minus sign)
user	user name as it is authenticated, or else it is - (a minus sign). If the user name contains spaces, they will be replaced by _ (an underscore).
	DD: day, MMM: a 3-letter abbreviation for the month name
DD/MMM/YYYY:HH:MM:SS(Jan, Feb,...), YYYY: year, HH: hour, MM: minutes, SS: seconds.	The date and time are local to the server.
request	request sent by the client (ex. GET /index.htm HTTP/1.0)
state	reply given by the server
length	size of the data returned (except the HTTP header) or 0
Referer	DLF only- Contains the URL of the page pointing to the requested document.
User-agent	DLF only- Contains the name and version of the browser or software of the client at the origin of the request

## **ELF/WLF**

The ELF (Extended Log Format) format is very widespread in the world of HTTP browsers. It can be used to build sophisticated logs that meet specific needs. For this reason, the ELF format can be customized: it is possible to choose the fields to be recorded as well as their order of insertion into the file.

The WLF (WebStar Log Format) was developed specifically for the 4D WebSTAR server.

## **Configuring the fields**

When you choose the ELF or WLF format, the "Web Log Token Selection" area displays the fields available for the chosen format. You will need to select each field to be included in the log. To do so, check the desired fields.

You cannot select the same field twice.

The following table lists the fields available for each format (in alphabetical order) and describes its contents:

<b>Field</b>	<b>ELFWLF</b>	<b>Value</b>
BYTES_RECEIVED	X	Number of bytes received by the server
BYTES_SENT	X X	Number of bytes sent by the server to the client
C_DNS	X X	IP address of the DNS (ELF: field identical to the C_IP field)
C_IP	X X	IP address of the client (for example 192.100.100.10)
CONNECTION_ID	X	Connection ID number
CS(COOKIE)	X X	Information about cookies contained in the HTTP request
CS(HOST)	X X	Host field of the HTTP request
CS(REFERER)	X X	URL of the page pointing to the requested document
CS(USER_AGENT)	X X	Information about the software and operating system of the client
CS_SIP	X X	IP address of the server
CS_URI	X X	URI on which the request is made
CS_URI_QUERY	X X	Request query parameters
CS_URI_STEM	X X	Part of request without query parameters
DATE	X X	DD: day, MMM: 3-letter abbreviation for month (Jan, Feb, etc.), YYYY: year
METHOD	X X	HTTP method used for the request sent to the server
PATH_ARGS	X	CGI parameters: string located after the "\$" character
STATUS	X X	Reply provided by the server
TIME	X X	HH: hour, MM: minutes, SS: seconds
TRANSFER_TIME	X X	Time requested by server to generate the reply
USER	X X	User name if authenticated; otherwise - (minus sign). If the user name contains spaces, they are replaced by _ (underlines)
URL	X	URL requested by the client

Dates and times are given in GMT.

## Backup Frequency

Since a *logweb.txt* file can become considerably large, it is possible to set up an automatic archiving mechanism. The triggering of a backup can be based on a certain period of time (expressed in hours, days, week or months), or based on the file size; when the set deadline (or file size) is reached, 4D automatically closes and archives the current log file and creates a new one.

When the web log file backup is triggered, the log file is archived in a folder named "Logweb Archives," which is created at the same level as the *logweb.txt* file.

The archived file is renamed based on the following example:

"YYYY\_MM\_DD\_Thh\_mm\_ss.txt." For instance, for a file archived on September 4, 2020 at 3:50 p.m. and 7 seconds: "D2020\_09\_04\_T15\_50\_07.txt."

## Backup Parameters

The automatic backup parameters for the *logweb.txt* are set on the **Web/Log (backup)** page of the Settings:

First you must choose the frequency (days, weeks, etc.) or the file size limit criterion by clicking on the corresponding radio button. You must then specify the precise moment of the backup if necessary.

- **No Backup:** The scheduled backup function is deactivated.
- **Every X hour(s):** This option is used to program backups on an hourly basis. You can enter a value between 1 and 24 .
  - **starting at:** Used to set the time at which the first back up will begin.
- **Every X day(s) at X:** This option is used to program backups on a daily basis. Enter 1 if you want to perform a daily backup. When this option is checked, you must indicate the time when the backup must be started.
- **Every X week(s), day at X:** This option is used to program backups on a weekly basis. Enter 1 if you want to perform a weekly backup. When this option is checked, you must indicate the day(s) of the week and the time when each backup must be started. You can select several days of the week if desired. For example, you can use this option to set two weekly backups: one on Wednesdays and one on Fridays.
- **Every X month(s), Xth day at X:** This option is used to program backups on a monthly basis. Enter 1 if you want to perform a monthly backup. When this option is checked, you must indicate the day of the month and the time when the backup must be started.
- **Every X MB:** This option is used to program backups based on the size of the current request log file. A backup is automatically triggered when the file reaches the set size. You can set a size limit of 1, 10, 100 or 1000 MB.

## Web Server object

A 4D project can start and monitor a web server for the main (host) application as well as each hosted component.

For example, if you installed two components in your main application, you can start and monitor up to three independant web servers from your application:

- one web server for the host application,
- one web server for the component #1,
- one web server for the component #2.

Other than memory, there is no limit to the number of components and thus, of web servers, that can be attached to a single 4D application project.

Each 4D web server, including the main application's web server, is exposed as a specific **object** of the `4D.WebServer` class. Once instantiated, a web server object can be handled from the current application or from any component using a [large number of properties and functions](#).

The legacy [WEB commands](#) of the 4D language are supported but cannot select the web server to which they apply (see below).

Each web server (host application or component) can be used in its own separate context, including:

- `On Web Authentication` and `On Web Connection` database method calls
- 4D tags processing and method calls,
- web sessions and TLS protocol management.

This allows you to develop independant components and features that come with their own web interfaces.

## Instantiating a web server object

The web server object of the host application (default web server) is automatically loaded by 4D at startup. Thus, if you write in a newly created project:

```
$nbSrv:=WEB Server list.length  
//$nbSrv value is 1
```

To instantiate a web server object, call the [WEB Server](#) command:

```
//create an object variable of the 4D.WebServer class  
var webServer : 4D.WebServer  
    //call the web server from the current context  
webServer:=WEB Server  
  
    //equivalent to  
webServer:=WEB Server(Web server database)
```

If the application uses components and you want to call:

- the host application's web server from a component or
- the server that received the request (whatever the server),

you can also use:

```

var webServer : 4D.WebServer
    //call the host web server from a component
webServer:=WEB Server(Web server host database)
    //call the target web server
webServer:=WEB Server(Web server receiving request)

```

## Web server functions

A [web server class object](#) contains the following functions:

<b>Functions</b>	<b>Parameter</b>	<b>Return value</b>	<b>Description</b>
<a href="#">start()</a>	settings (object)	status (object)	Starts the web server
<a href="#">stop()</a>	-	-	Stops the web server

To start and stop a web server, just call the [start\(\)](#) and [stop\(\)](#) functions of the web server object:

```

var $status : Object
    //to start a web server with default settings
$status:=webServer.start()
    //to start the web server with custom settings
    //settings object contains web server properties
webServer.start($settings)

    //to stop the web server
$status:=webServer.stop()

```

## Web server properties

A web server object contains [various properties](#) which configure the web server.

These properties are defined:

1. using the `settings` parameter of the [.start\(\)](#) function (except for read-only properties, see below),
2. if not used, using the `WEB SET OPTION` command (host applications only),
3. if not used, in the settings of the host application or the component.
  - If the web server is not started, the properties contain the values that will be used at the next web server startup.
  - If the web server is started, the properties contain the actual values used by the web server (default settings could have been overridden by the `settings` parameter of the [.start\(\)](#) function).

`isRunning`, `name`, `openSSLVersion`, and `perfectForwardSecrecy` are read-only properties that cannot be predefined in the `settings` object parameter for the [start\(\)](#) function.

## Scope of the 4D Web commands

The 4D Language contains [several commands](#) that can be used to control the web server. However, these commands are designed to work with a single (default) web server. When using these commands in the context of web server objects, make sure their scope is appropriate.

<b>Command</b>	<b>Scope</b>
SET DATABASE PARAMETER	Host application web server
WEB CLOSE SESSION	Web server that received the request
WEB GET BODY PART	Web server that received the request
WEB Get body part count	Web server that received the request
WEB Get Current Session ID	Web server that received the request
WEB GET HTTP BODY	Web server that received the request
WEB GET HTTP HEADER	Web server that received the request
WEB GET OPTION	Host application web server
WEB Get server info	Host application web server
WEB GET SESSION EXPIRATION	Web server that received the request
WEB Get session process count	Web server that received the request
WEB GET STATISTICS	Host application web server
WEB GET VARIABLES	Web server that received the request
WEB Is secured connection	Web server that received the request
WEB Is server running	Host application web server
WEB SEND BLOB	Web server that received the request
WEB SEND FILE	Web server that received the request
WEB SEND HTTP REDIRECT	Web server that received the request
WEB SEND RAW DATA	Web server that received the request
WEB SEND TEXT	Web server that received the request
WEB SET HOME PAGE	Host application web server
WEB SET HTTP HEADER	Web server that received the request
WEB SET OPTION	Host application web server
WEB SET ROOT FOLDER	Host application web server
WEB START SERVER	Host application web server
WEB STOP SERVER	Host application web server
WEB Validate digest	Web server that received the request

# Web Development

This "Getting started" section is geared at first-time users who want an overall overview on how to go from zero to a 4D website that handles data from the database. Let's start!

## Hello World Example

Let's start by making the web server send "Hello World" to the browser. The most simple way to do this is to create a project, start the web server and write a small code that returns a text in the `On Web Connection` database method.

### Starting the web server

To start the 4D web server:

1. Launch your 4D application and create a new, empty 4D project.
2. In the **Run** menu, select **Start Web Server**.

That's all! The web server is started (you can see that the menu item has become **Stop Web Server**). It is now ready to handle requests. To check it, we'll display the default home page.

### Displaying the default home page

The 4D web server creates automatically a default `index.html` page in the default `WebFolder` root folder, created at the same level as the Project folder.

1. Launch a web browser and connect to the web server IP address (default http port for 4D web server is 80). If the web server and the browser are on the same machine, you can select **Test Web Server** in the **Run** menu.

The default home page is displayed:

### Displaying Hello World

1. Open the Explorer, display the Database Methods list and double-click on `On Web Connection`.
2. Enter the following code:

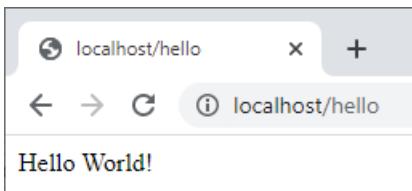
```
Case of
  : ($1="/hello")
    WEB SEND TEXT("Hello World!")
  Else
    // Error 404 for example
End case
```

The `On Web Connection` database method is called for incoming requests and receives the target URL in the `$1` parameter. This very simple code only sends the text to the browser.

3. In your browser, enter the following URL:

`http://localhost/hello`

The web server handles the request and returns:



## Getting data from the database

Now we'll see how simple it is to get data from the database. First, we will create a table and fill it with some data.

Create a basic database with, for example, a single table containing some records:

A screenshot of a database application. On the left, there is a table named 'Friends' with columns 'ID', 'lastName', and 'firstName'. It contains 4 records. On the right, there is a window titled 'Hello World' showing a table with 'First Name' and 'Last Name' columns, also containing 4 records. Both tables show the same data: John Smith, Danny Brown, Mark Purple, and Jenny Dupont.

	First Name	Last Name
ID	John	Smith
lastName	Danny	Brown
firstName	Mark	Purple
	Jenny	Dupont

## Displaying data in a page

The most simple solution to display data is to call a [template page](#) containing tags.

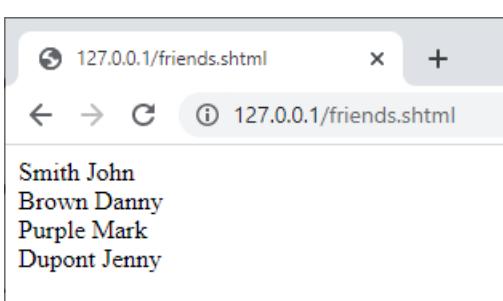
1. Using any text editor, create a file containing the following lines:

```
<html>
<body>
<!--#4DCODE ALL RECORDS ([Friends])-->
<!--#4DLOOP [Friends]-->
<!--#4DTEXT [Friends]lastName--> <!--#4DTEXT [Friends]firstName--
><br/>
<!--#4DENDLOOP-->
</body>
</html>
```

2. Name the file "friends.shtml" and save it in the **WebFolder** of your project.
3. In your browser, enter the following URL:

<http://localhost/friends.shtml>

.shtml pages are automatically processed by the web server. Your page filled with data is returned:



## REST request

If we not only want to *display* data, but to *use* it, we can use ORDA and the REST server. Thanks to the [ORDA concept](#), the `Friends` table is automatically mapped to a dataclass and is available through [REST](#).

1. We will use the REST server to access data: go the **Settings** dialog box, select **Web > Web Features**, and check the **Expose as REST server** option.

2. In your browser, enter the following URL:

```
http://localhost/rest/$catalog
```

The web server returns the results in JSON:

```
{  
    "__UNIQID": "3F1B6ACFFE12B64493629AD76011922D",  
    "dataClasses": [  
        {  
            "name": "Friends",  
            "uri": "/rest/$catalog/Friends",  
            "dataURI": "/rest/Friends"  
        }  
    ]  
}
```

You get the catalog, i.e. the list of exposed dataclasses and attributes in the datastore.

You can also get any data.

3. Enter the following URL:

```
http://localhost/rest/Friends
```

The server returns the entities, i.e. the data, from the Friends dataclass:

```

{
    "__DATACLASS": "Friends",
    "__entityModel": "Friends",
    "__GlobalStamp": 0,
    "__COUNT": 4,
    "__FIRST": 0,
    "__ENTITIES": [
        {
            "__KEY": "1",
            "__TIMESTAMP": "2020-10-27T14:29:01.914Z",
            "__STAMP": 1,
            "ID": 1,
            "lastName": "Smith",
            "firstName": "John"
        },
        {
            "__KEY": "2",
            "__TIMESTAMP": "2020-10-27T14:29:16.035Z",
            "__STAMP": 1,
            "ID": 2,
            "lastName": "Brown",
            "firstName": "Danny"
        },
        {
            "__KEY": "3",
            "__TIMESTAMP": "2020-10-27T14:29:43.945Z",
            "__STAMP": 1,
            "ID": 3,
            "lastName": "Purple",
            "firstName": "Mark"
        },
        {
            "__KEY": "4",
            "__TIMESTAMP": "2020-10-27T14:34:58.457Z",
            "__STAMP": 1,
            "ID": 4,
            "lastName": "Dupont",
            "firstName": "Jenny"
        }
    ],
    "__SENT": 4
}

```

This very simple example shows how the web server interacts transparently with the [REST server](#) to return any requested data, provided it is exposed. In your web interfaces, you can easily bind the javascript or html code with returned data. See the built-in [Web Data Explorer](#) to have an example of sophisticated web interface bound to dataclasses.

## Login and session

In the above sections, we get free access to the application from web requests. However, in the world of web applications, data access security is the first priority. When connecting to the 4D web server, users must be authentified and their navigation controlled.

### Creating a table of users

The most simple and secured way to log a user on the 4D web server is based upon the following scenario:

- Users are stored in a dedicated, unexposed table (named *WebUsers* for example)
- The *WebUsers* table could be [encrypted](#) and stores the user login and a hash of

their password.

1. Create a table with some fields, for example:

WebUsers	
ID	2 <sup>32</sup>
userId	A
lastName	A
password	A
firstName	A

2. Write and execute the following code to create a user:

```
var $webUser : cs.WebUsersEntity  
  
$webUser:=ds.WebUsers.new()  
$webUser.firstName:="John"  
$webUser.lastName:="Doe"  
// the password would be entered by the user  
$webUser.password:=Generate password hash("123")  
$webUser.userId:="john@4d.com"  
$webUser.save()
```

## Authenticating users

To be secure from end to end, it is necessary that the whole connection is established via [https](https://).

1. Open the Explorer and create a project method named "login".
2. Write the following code:

```

var $indexUserId; $indexPassword : Integer
var $userId; $password : Text
var $user; $info : Object
ARRAY TEXT($anames; 0)
ARRAY TEXT($avalues; 0)

// get values sent in the header of the request
WEB GET VARIABLES($anames; $avalues)

// look for header login fields
$indexUserId:=Find in array($anames; "userId")
$userId:=$avalues{$indexUserId}
$indexPassword:=Find in array($anames; "password")
$password:=$avalues{$indexPassword}

//look for a user with the entered name in the users table
$user:=ds.WebUsers.query("userId = :1"; $userId).first()

If ($user#Null) //a user was found
    //check the password
    If (Verify password hash($password; $user.password))
        //password ok, fill the session
        $info:=New object()
        $info.userName:=$user.firstName+" "+$user.lastName
        Session.setPrivileges($info)
        //You can use the user session to store any information
        WEB SEND TEXT("Welcome "+Session.userName)
    Else
        WEB SEND TEXT("Wrong user name or password.")
    End if
Else
    WEB SEND TEXT("Wrong user name or password.")
End if

```

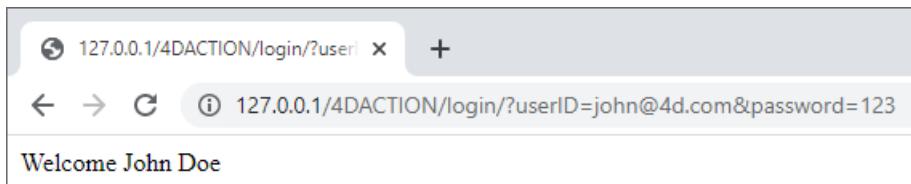
3. Display the method properties by clicking on the **[i]** button in the code editor, check the **4D tags and URLs (4DACTION...)** option and click **OK**.

4. In your browser, enter the following URL:

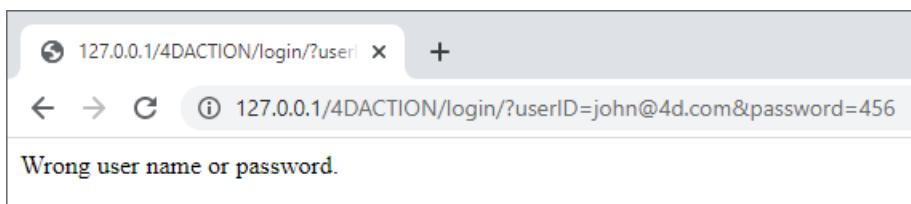
<http://localhost/4DACTION/login/?userID=john@4d.com&password=123>

Using such URLs is not recommended, it is only presented here to keep the example simple. A more realistic login request must be handled through a web form and a POST request. See [this page](#) for an example of form POST.

Then you will be logged for the session:



Wrong credentials would be rejected:



Once a user is logged, you can handle the associated session using the `WEB Get Current Session ID` method. See the [User sessions](#) page.

## Template pages

4D's Web server allows you to use HTML template pages containing tags, i.e. a mix of static HTML code and 4D references added by means of [transformation tags](#) such as 4DTEXT, 4DIF, or 4DINCLUDE. These tags are usually inserted as HTML type comments (<!--#4DTagName TagValue--> ) into the HTML source code.

When these pages are sent by the HTTP server, they are parsed and the tags they contain are executed and replaced with the resulting data. The pages received by the browsers are thus a combination of static elements and values coming from 4D processing.

For example, if you write in an HTML page:

```
<P>Welcome to <!--#4DTEXT vtSiteName--!</P>
```

The value of the 4D variable *vtSiteName* will be inserted in the HTML page.

## Tags for templates

The following 4D tags are available:

- 4DTEXT, to insert 4D variables and expressions as text,
- 4DHTML, to insert HTML code,
- 4DEVAL, to evaluate any 4D expression,
- 4DSCRIPT, to execute a 4D method,
- 4DINCLUDE, to include a page within another one,
- 4DBASE, to modify the default folder used by the 4DINCLUDE tag,
- 4DCODE, to insert 4D code,
- 4DIF, 4ELSE, 4ELSEIF and 4ENDIF, to insert conditions in the HTML code,
- 4DOLOOP and 4DENDLOOP, to make loops in the HTML code,
- 4DEACH and 4DENDEACH, to loop in collections, entity selections, or object properties.

These tags are described in the [Transformation Tags](#) page.

It is possible to mix tags. For example, the following HTML code is allowed:

```

<HTML>
...
<BODY>
<!--#4DSCRIPT/PRE_PROCESS-->      (Method call)
<!--#4DIF (myvar=1)-->      (If condition)
    <!--#4DINCLUDE banner1.html-->      (Subpage insertion)
<!--#4DENDIF-->      (End if)
<!--#4DIF (myvar=2)-->

    <!--#4DINCLUDE banner2.html-->
<!--#4DENDIF-->

<!--#4DLOOP [TABLE]-->      (loop on the current selection)
<!--#4DIF ([TABLE]ValNum>10)-->      (If [TABLE]ValNum>10)
    <!--#4DINCLUDE subpage.html-->      (subpage insertion)
<!--#4DELSE-->      (Else)
    <B>Value: <!--#4DTEXT [TABLE]ValNum--></B><br />
        (Field display)
<!--#4DENDIF-->
<!--#4DENDLOOP-->      (End for)
</BODY>
</HTML>

```

## Tag parsing

For optimization reasons, the parsing of the HTML source code is not carried out by the 4D Web server when HTML pages are called using simple URLs that are suffixed with ".HTML" or ".HTM".

Parsing of the contents of template pages sent by 4D web server takes place when `WEB SEND FILE` (.htm, .html, .shtm, .shtml), `WEB SEND BLOB` (text/html type BLOB) or `WEB SEND TEXT` commands are called, as well as when sending pages called using URLs. In this last case, for reasons of optimization, pages that are suffixed with ".htm" and ".html" are NOT parsed. In order to "force" the parsing of HTML pages in this case, you must add the suffix ".shtm" or ".shtml" (for example, <http://www.server.com/dir/page.shtm>). An example of the use of this type of page is given in the description of the `WEB GET STATISTICS` command. XML pages (.xml, .xsl) are also supported and always parsed by 4D.

You can also carry out parsing outside of the Web context when you use the `PROCESS 4D TAGS` command.

Internally, the parser works with UTF-16 strings, but the data to parse may have been encoded differently. When tags contain text (for example `4DHTML`), 4D converts the data when necessary depending on its origin and the information available (charset). Below are the cases where 4D parses the tags contained in the HTML pages, as well as any conversions carried out:

Action / Command	Content analysis of the sent pages	Support of \$ syntax(*)	Character set used for parsing tags
Pages called via URLs	X, except for pages with ".htm" or ".html" extensions	X, except for pages with ".htm" or ".html" extensions	Use of charset passed as parameter of the "Content-Type" header of the page. If there is none, search for a META-HTTP EQUIV tag with a charset. Otherwise, use of default character set for the HTTP server
WEB SEND FILE	X	-	Use of charset passed as parameter of the "Content-Type" header of the page. If there is none, search for a META-HTTP EQUIV tag with a charset. Otherwise, use of default character set for the HTTP server
WEB SEND TEXT	X	-	No conversion necessary
WEB SEND BLOB	X, if BLOB is of the "text/html" type	-	Use of charset set in the "Content-Type" header of the response. Otherwise, use of default character set for the HTTP server
Inclusion by the <!-- #4DINCLUDE--> tag	X	X	Use of charset passed as parameter of the "Content-Type" header of the page. If there is none, search for a META-HTTP EQUIV tag with a charset. Otherwise, use of default character set for the HTTP server
PROCESS 4D TAGS	X	X	Text data: no conversion. BLOB data: automatic conversion from the Mac-Roman character set for compatibility

(\*) The alternative \$-based syntax is available for 4DHTML, 4DTEXT and 4DEVAL tags.

## Accessing 4D methods via the Web

Executing a 4D method with `4DEACH`, `4DELSEIF`, `4DEVAL`, `4DHTML`, `4DIF`, `4DLOOP`, `4SCRIPT`, or `4DTEXT` from a web request is subject to the [Available through 4D tags and URLs \(4ACTION...\)](#) attribute value defined in the properties of the method. If the attribute is not checked for the method, it can not be called from a web request.

## Prevention of malicious code insertion

4D tags accept different types of data as parameters: text, variables, methods, command names, etc. When this data is provided by your own code, there is no risk of malicious code insertion since you control the input. However, your database code often works with data that was, at one time or another, introduced through an external source (user input, import, etc.).

In this case, it is advisable to **not use** tags such as `4DEVAL` or `4SCRIPT`, which evaluate parameters, directly with this sort of data.

In addition, according to the [principle of recursion](#), malicious code may itself include transformation tags. In this case, it is imperative to use the `4DTEXT` tag. Imagine, for example, a Web form field named "Name", where users must enter their name. This name is then displayed using a `<!--#4DHTML vName-->` tag in the page. If text of the "\$%" type is inserted instead of the name, interpreting this tag will cause the application to be exited. To avoid this risk, you can just use the `4DTEXT` tag systematically in this case. Since this tag escapes the special HTML characters, any malicious recursive code

that may have been inserted will not be reinterpreted. To refer to the previous example, the "Name" field will contain, in this case, "<!#4DEVAL QUIT 4D-->" which will not be transformed.

# Processing HTTP requests

The 4D web server provides several features to handle HTTP requests:

- the `On Web Connection` database method, a router for your web application,
- the `/4DACTION` URL to call server-side code
- `WEB GET VARIABLES` to get values from HTML objects sent to the server
- other commands such as `WEB GET HTTP BODY`, `WEB GET HTTP HEADER`, or `WEB GET BODY PART` allow to customize the request processing, including cookies.
- the `COMPILER_WEB` project method, to declare your variables.

## On Web Connection

The `On Web Connection` database method can be used as the entry point for the 4D Web server.

### Database method calls

The `On Web Connection` database method is automatically called when the server receives any URL that is not a path to an existing page on the server. The database method is called with the URL.

For example, the URL "a/b/c" will call the database method, but "a/b/c.html" will not call the database method if the page "c.html" exists in the "a/b" subfolder of the [WebFolder](#).

The request should have previously been accepted by the [On Web Authentication](#) database method (if it exists) and the web server must be launched.

### Syntax

**On Web Connection( \$1 : Text ; \$2 : Text ; \$3 : Text ; \$4 : Text ; \$5 : Text ; \$6 : Text )**

Parameters	Type	Description
\$1	Text <-URL	
\$2	Text <- HTTP headers + HTTP body (up to 32 kb limit)	
\$3	Text <-IP address of the web client (browser)	
\$4	Text <-IP address of the server	
\$5	Text <-User name	
\$6	Text <-Password	

You must declare these parameters as shown below:

```
//On Web Connection database method  
C_TEXT($1;$2;$3;$4;$5;$6)  
//Code for the method
```

Alternatively, you can use the [named parameters](#) syntax:

```
// On Web Connection database method
#DECLARE ($url : Text; $header : Text; \
$BrowserIP : Text; $ServerIP : Text; \
$user : Text; $password : Text)
```

Calling a 4D command that displays an interface element (`DIALOG`, `ALERT`, etc.) is not allowed and ends the method processing.

## \$1 - URL extra data

The first parameter (\$1) is the URL entered by users in the address area of their web browser, without the host address.

Let's use an intranet connection as an example. Suppose that the IP address of your 4D Web Server machine is 123.4.567.89. The following table shows the values of \$1 depending on the URL entered in the web browser:

URL entered in web browser	Value of parameter \$1
123.4.567.89	/
<a href="http://123.4.567.89">http://123.4.567.89</a>	/
123.4.567.89/Customers	/Customers
<a href="http://123.4.567.89/Customers/Add">http://123.4.567.89/Customers/Add</a>	/Customers/Add
123.4.567.89/Do_This/If_OK/Do_That/Do_This/If_OK/Do_That	

Note that you are free to use this parameter at your convenience. 4D simply ignores the value passed beyond the host part of the URL. For example, you can establish a convention where the value "/Customers/Add" means "go directly to add a new record in the `[Customers]` table." By supplying the web users with a list of possible values and/or default bookmarks, you can provide shortcuts to different parts of your application. This way, web users can quickly access resources of your website without going through the entire navigation path each time they make a new connection.

## \$2 - Header and Body of the HTTP request

The second parameter (\$2) is the header and the body of the HTTP request sent by the web browser. Note that this information is passed to your `On Web Connection` database method "as is". Its contents will vary depending on the nature of the web browser attempting the connection.

If your application uses this information, it is up to you to parse the header and the body. You can use the `WEB GET HTTP HEADER` and the `WEB GET HTTP BODY` commands.

For performance reasons, the size of data passing through the \$2 parameter must not exceed 32 KB. Beyond this size, they are truncated by the 4D HTTP server.

## \$3 - Web client IP address

The \$3 parameter receives the IP address of the browser's machine. This information can allow you to distinguish between intranet and internet connections.

4D returns IPv4 addresses in a hybrid IPv6/IPv4 format written with a 96-bit prefix, for example ::ffff:192.168.2.34 for the IPv4 address 192.168.2.34. For more information, refer to the [IPv6 Support](#) section.

## \$4 - Server IP address

The \$4 parameter receives the IP address requested by the 4D Web Server. 4D allows for multi-homing, which allows you to use machines with more than one IP address. For more information, please refer to the [Configuration page](#).

## \$5 and \$6 - User Name and Password

The \$5 and \$6 parameters receive the user name and password entered by the user in the standard identification dialog box displayed by the browser, if applicable (see the [authentication page](#)).

If the user name sent by the browser exists in 4D, the \$6 parameter (the user's password) is not returned for security reasons.

## /4DACTION

**/4DACTION/\*MethodName\***  
**/4DACTION/\*MethodName/Param\***

<b>Parameters Type</b>	<b>Description</b>
MethodNameText ->	Name of the 4D project method to be executed
Param	Text -> Text parameter to pass to the project method

**Usage:** URL or Form action.

This URL allows you to call the *MethodName* 4D project method with an optional *Param* text parameter. The method will receive this parameter in \$1.

- The 4D project method must have been [allowed for web requests](#): the "Available through 4D tags and URLs (4DACTION...)" attribute value must have been checked in the properties of the method. If the attribute is not checked, the web request is rejected.
- When 4D receives a /4DACTION/MethodName/Param request, the [On Web Authentication](#) database method (if it exists) is called.

4DACTION/ can be associated with a URL in a static Web page:

```
<A HREF="/4DACTION/MyMethod/hello">Do Something</A>
```

The [MyMethod](#) project method should generally return a "reply" (sending of an HTML page using [WEB SEND FILE](#) or [WEB SEND TEXT](#), etc.). Be sure to make the processing as short as possible in order not to block the browser.

A method called by [/4DACTION](#) must not call interface elements ([DIALOG](#), [ALERT](#), etc.).

### Example

This example describes the association of the [/4DACTION](#) URL with an HTML picture object in order to dynamically display a picture in the page. You insert the following instructions in a static HTML page:

```
<IMG SRC="/4DACTION/getPhoto/smith">
```

The [getPhoto](#) method is as follows:

```

C_TEXT($1) // This parameter must always be declared
var $path : Text
var $PictVar : Picture
var $BlobVar : Blob

//find the picture in the Images folder within the Resources folder
$path:=Get 4D folder(Current resources folder)+"Images"+Folder
separator+$1+".psd"

READ PICTURE FILE($path;$PictVar) //put the picture in the picture
variable
PICTURE TO BLOB($PictVar;$BLOB;".png") //convert the picture to ".png"
format
WEB SEND BLOB ($BLOB;"image/png")

```

## 4ACTION to post forms

The 4D Web server also allows you to use “posted” forms, which are static HTML pages that send data to the Web server, and to easily retrieve all the values. The POST type must be associated to them and the form’s action must imperatively start with /4ACTION/MethodName.

A form can be submitted through two methods (both can be used with 4D):

- POST, usually used to send data to the Web server,
- GET, usually used to request data from the Web server.

When the Web server receives a posted form, it calls the `On Web Authentication` database method (if it exists).

In the called method, you must call the `WEB GET VARIABLES` command in order to [retrieve the names and values](#) of all the fields included in an HTML page submitted to the server.

Example to define the action of a form:

```
<FORM ACTION="/4ACTION/MethodName" METHOD=POST>
```

### Example

In a Web application, we would like for the browsers to be able to search among the records by using a static HTML page. This page is called “search.htm”. The application contains other static pages that allow you to, for example, display the search result (“results.htm”). The POST type has been associated to the page, as well as the `/4ACTION/SEARCH` action.

Here is the HTML code that corresponds to this page:

```
<form action="/4action/processForm" method=POST>
<input type=text name=vName value=""><br />
<input type=checkbox name=vExact value="Word">Whole word<br />
<input type=submit name=OK value="Search">
</FORM>
```

During data entry, type “ABCD” in the data entry area, check the “Whole word” option and validate it by clicking the **Search** button. In the request sent to the Web server:

```
vName="ABCD"
vExact="Word"
OK="Search"
```

4D calls the `On Web Authentication` database method (if it exists), then the

`processForm` project method is called, which is as follows:

```
C_TEXT($1) //mandatory for compiled mode
C_LONGINT($vName)
C_TEXT(vName;vLIST)
ARRAY TEXT($arrNames;0)
ARRAY TEXT($arrVals;0)
WEB GET VARIABLES($arrNames;$arrVals) //we retrieve all the variables
of the form
$vName:=Find in array($arrNames;"vName")
vName:=$arrVals{$vName}
If(Find in array($arrNames;"vExact")=-1) //If the option has not been
checked
    vName:=vName+"@"
End if
QUERY([Jockeys];[Jockeys]Name=vName)
FIRST RECORD([Jockeys])
While(Not(End selection([Jockeys])))
    vLIST:=vLIST+[Jockeys]Name+" "+[Jockeys]Tel+<br/>
    NEXT RECORD([Jockeys])
End while
WEB SEND FILE("results.htm") //Send the list to the results.htm form
//which contains a reference to the variable vLIST,
//for example <!--4DHTML vLIST-->
//...
End if
```

## Getting values from HTTP requests

4D's Web server lets you recover data sent through POST or GET requests, using Web forms or URLs.

When the Web server receives a request with data in the header or in the URL, 4D can retrieve the values of any HTML objects it contains. This principle can be implemented in the case of a Web form, sent for example using `WEB SEND FILE` or `WEB SEND BLOB`, where the user enters or modifies values, then clicks on the validation button.

In this case, 4D can retrieve the values of the HTML objects found in the request using the `WEB GET VARIABLES` command. The `WEB GET VARIABLES` command retrieves the values as text.

Consider the following HTML page source code:

```

<html>
<head>
    <title>Welcome</title>
    <script language="JavaScript"><!--
function GetBrowserInformation(formObj) {
formObj.vtNav_appName.value = navigator.appName
formObj.vtNav_appVersion.value = navigator.appVersion
formObj.vtNav_appCodeName.value = navigator.appCodeName
formObj.vtNav_userAgent.value = navigator.userAgent
return true
}
function LogOn(formObj) {
if(formObj.vtUserName.value!="") {
return true
} else {
alert("Enter your name, then try again.")
return false
}
}
//--></script>
</head>
<body>
<form action="/4DACTION/WWW_STD_FORM_POST" method="post"
name="frmWelcome"
onsubmit="return GetBrowserInformation(frmWelcome)">
    <h1>Welcome to Spiders United</h1>
    <p><b>Please enter your name:</b>
        <input name="vtUserName" value="" size="30" type="text"></p>
    <p>
        <input name="vsbLogOn" value="Log On" onclick="return
LogOn(frmWelcome)" type="submit">
        <input name="vsbRegister" value="Register" type="submit">
        <input name="vsbInformation" value="Information" type="submit"></p>
    <p>
        <input name="vtNav_appName" value="" type="hidden">
        <input name="vtNav_appVersion" value="" type="hidden">
        <input name="vtNav_appCodeName" value="" type="hidden">
        <input name="vtNav_userAgent" value="" type="hidden"></p>
    </form>
</body>
</html>

```

When 4D sends the page to a Web Browser, it looks like this:

The main features of this page are:

- It includes three **Submit** buttons: `vsbLogOn`, `vsbRegister` and `vsbInformation`.
- When you click **Log On**, the submission of the form is first processed by the JavaScript function `LogOn`. If no name is entered, the form is not even submitted to 4D, and a JavaScript alert is displayed.
- The form has a POST 4D method as well as a Submit script (`GetBrowserInformation`) that copies the browser properties to the four hidden objects whose names starts with `vtNav_App`. It also includes the `vtUserName` object.

Let's examine the 4D method `WWW_STD_FORM_POST` that is called when the user clicks on one of the buttons on the HTML form.

```

// Retrieval of value of variables
ARRAY TEXT($arrNames;0)
ARRAY TEXT($arrValues;0)
WEB GET VARIABLES($arrNames;$arrValues)
C_TEXT($user)

Case of

// The Log On button was clicked
:(Find in array($arrNames;"vsbLogOn") #-1)
$user :=Find in array($arrNames;"vtUserName")
QUERY([WWW Users];[WWW Users]UserName=$arrValues{$user})
$0:=(Records in selection([WWW Users])>0)
If($0)
    WWW POST EVENT("Log On";WWW Log information)
// The WWW POST EVENT method saves the information in a database
table
Else

    $0:=WWW Register
// The WWW Register method lets a new Web user register
End if

// The Register button was clicked
:(Find in array($arrNames;"vsbRegister") #-1)
$0:=WWW Register

// The Information button was clicked
:(Find in array($arrNames;"vsbInformation") #-1)
WEB SEND FILE("userinfos.html")
End case

```

The features of this method are:

- The values of the variables *vtNav\_appName*, *vtNav\_appVersion*, *vtNav\_appCodeName*, and *vtNav\_userAgent* (bound to the HTML objects having the same names) are retrieved using the `WEB GET VARIABLES` command from HTML objects created by the *GetBrowserInformation* JavaScript script.
- Out of the *vsbLogOn*, *vsbRegister* and *vsbInformation* variables bound to the three Submit buttons, only the one corresponding to the button that was clicked will be retrieved by the `WEB GET VARIABLES` command. When the submit is performed by one of these buttons, the browser returns the value of the clicked button to 4D. This tells you which button was clicked.

Keep in mind that with HTML, all objects are text objects. If you use a SELECT object, it is the value of the highlighted element in the object that is returned in the `WEB GET VARIABLES` command, and not the position of the element in the array as in 4D. `WEB GET VARIABLES` always returns values of the Text type.

## Other Web Server Commands

The 4D web server provides several low-level web commands allowing you to develop custom processing of requests:

- the `WEB GET HTTP BODY` command returns the body as raw text, allowing any parsing you may need
- the `WEB GET HTTP HEADER` command return the headers of the request. It is useful to handle custom cookies, for example (along with the `WEB SET HTTP HEADER` command).
- the `WEB GET BODY PART` and `WEB Get body part count` commands to parse

the body part of a multi-part request and retrieve text values, but also files posted, using BLOBs.

These commands are summarized in the following graphic:

The 4D web server supports files uploaded in chunked transfer encoding from any Web client. Chunked transfer encoding is a data transfer mechanism specified in HTTP/1.1. It allows data to be transferred in a series of "chunks" (parts) without knowing the final data size. The 4D Web Server also supports chunked transfer encoding from the server to Web clients (using `WEB SEND RAW DATA`).

## **COMPILER\_WEB Project Method**

The COMPILER\_WEB method, if it exists, is systematically called when the HTTP server receives a dynamic request and calls the 4D engine. This is the case, for example, when the 4D Web server receives a posted form or a URL to process in [On Web Connection](#). This method is intended to contain typing and/or variable initialization directives used during Web exchanges. It is used by the compiler when the application is compiled. The COMPILER\_WEB method is common to all the Web forms. By default, the COMPILER\_WEB method does not exist. You must explicitly create it.

The COMPILER\_WEB project method is also called, if it exists, for each SOAP request accepted.

## Allowing project methods

The 4D tags such as `4DEVAL`, `4DTEXT`, `4DHTML`... as well as the [/4ACTION URL](#) allow you to trigger the execution of any project method of a 4D project published on the Web. For example, the request <http://www.server.com/4ACTION/login> causes the execution of the **login** project method, if it exists.

This mechanism therefore presents a security risk for the application, in particular if an Internet user intentionally (or unintentionally) triggers a method not intended for execution via the web. You can avoid this risk in the following ways:

- Filter the methods called via the URLs using the [On Web Authentication](#) database method. Drawbacks: If the database includes a great number of methods, this system may be difficult to manage.
- Use the **Available through 4D tags and URLs (4ACTION...)** option found in the Method properties dialog box:

This option is used to individually designate each project method that can be called using the `4ACTION` special URL, or the `4DTEXT`, `4DHTML`, `4DEVAL`, `4DSCRIPT`, `4DIF`, `4ELSEIF` or `4DOOP` tags. When it is not checked, the project method concerned cannot be directly executed through an HTTP request. Conversely, it can be executed using other types of calls (formulas, other methods, etc.).

This option is unchecked by default. Methods that can be executed through `4ACTION` or specific tags must be specifically indicated.

In the Explorer, Project methods with this property are given a specific icon:



## Custom HTTP Error Pages

The 4D Web Server allows you to customize HTTP error pages sent to clients, based on the status code of the server response. Error pages refer to:

- status codes starting with 4 (client errors), for example 404
- status codes starting with 5 (server errors), for example 501.

For a full description of HTTP error status codes, you can refer to the [List of HTTP status codes](#) (Wikipedia).

## Replacing default pages

To replace default 4D Web Server error pages with your own pages you just need to:

- put custom HTML pages at the first level of the application's web folder,
- name the custom pages "{statusCode}.html" (for example, "404.html").

You can define one error page per status code and/or a generic error page for a range of errors, named "{number}xx.html". For example, you can create "4xx.html" for generic client errors. The 4D Web Server will first look for a {statusCode}.html page then, if it does not exist, a generic page.

For example, when an HTTP response returns a status code 404:

1. 4D Web Server tries to send a "404.html" page located in the application's web folder.
2. If it is not found, 4D Web Server tries to send a "4xx.html" page located in the application's web folder.
3. If not found, 4D Web Server then uses its default error page.

## Example

If you define the following custom pages in your web folder:

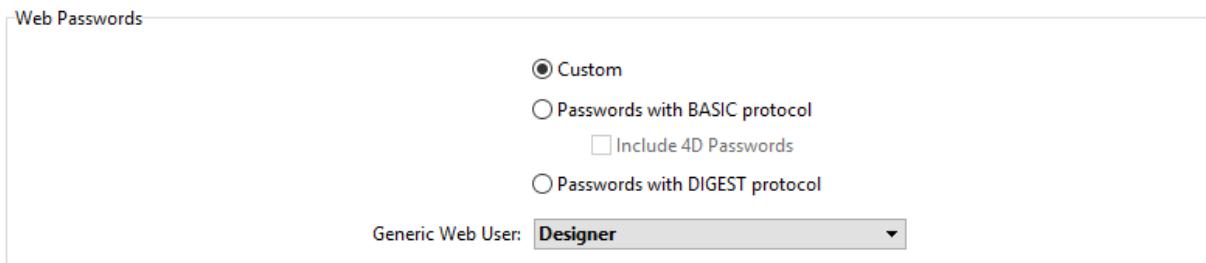
- the "403.html" or "404.html" pages will be served when 403 or 404 HTTP responses are returned respectively,
- the "4xx.html" page will be served for any other 4xx error status (400, 401, etc.),
- the "5xx.html" page will be served for any 5xx error status.

# Authentication

Authenticating users is necessary when you want to provide specific access rights to web users. Authentication designates the way the information concerning the user credentials (usually name and password) are collected and processed.

## Authentication modes

The 4D web server proposes three authentication modes, that you can select in the **Web/Options (I)** page of the Settings dialog box:



Using a **custom** authentication is recommended.

## Overview

The operation of the 4D web server's access system is summarized in the following diagram:

Requests starting with `rest/` are directly handled by the [REST server](#).

### Custom (default)

Basically in this mode, it's up to the developer to define how to authenticate users. 4D only evaluates HTTP requests [that require an authentication](#).

This authentication mode is the most flexible because it allows you to:

- either, delegate the user authentication to a third-party application (e.g. a social network, SSO);
- or, provide an interface to the user (e.g. a web form) so that they can create their account in your customer database; then, you can authenticate users with any custom algorithm (see [this example](#) from the "User sessions" chapter). The important thing is that you never store the password in clear, using such code:

```
//... user account creation  
ds.webUser.password:=Generate password hash($password)  
ds.webUser.save()
```

See also [this example](#) from the "Getting started" chapter.

If no custom authentication is provided, 4D calls the [On Web Authentication](#) database method (if it exists). In addition to \$1 and \$2, only the IP addresses of the browser and the server (\$3 and \$4) are provided, the user name and password (\$5 and \$6) are empty. The method must return **True** in \$0 if the user is successfully authenticated, then the requested resource is served, or **False** in \$0 if the authentication failed.

**Warning:** If the `On Web Authentication` database method does not exist, connections are automatically accepted (test mode).

## Basic protocol

When a user connects to the server, a standard dialog box appears on their browser in order for them to enter their user name and password.

The name and password entered by the user are sent unencrypted in the HTTP request header. This mode typically requires HTTPS to provide confidentiality.

Entered values are then evaluated:

- If the **Include 4D passwords** option is checked, user credentials will be first evaluated against the [internal 4D users table](#).
  - If the user name sent by the browser exists in the table of 4D users and the password is correct, the connection is accepted. If the password is incorrect, the connection is refused.
  - If the user name does not exist in the table of 4D users, the `On Web Authentication` database method is called. If the `On Web Authentication` database method does not exist, connections are rejected.
- If the **Include 4D passwords** option is not checked, user credentials are sent to the `On Web Authentication` database method along with the other connection parameters (IP address and port, URL...) so that you can process them. If the `On Web Authentication` database method does not exist, connections are rejected.

With the 4D Client web server, keep in mind that all the sites published by the 4D Client machines will share the same table of users. Validation of users/passwords is carried out by the 4D Server application.

## DIGEST protocol

This mode provides a greater level of security since the authentication information is processed by a one-way process called hashing which makes their contents impossible to decipher.

As in BASIC mode, users must enter their name and password when they connect. The `On Web Authentication` database method is then called. When the DIGEST mode is activated, the \$6 parameter (password) is always returned empty. In fact, when using this mode, this information does not pass by the network as clear text (unencrypted). It is therefore imperative in this case to evaluate connection requests using the `WEB Validate digest` command.

You must restart the web server in order for the changes made to these parameters to be taken into account.

## On Web Authentication

The `On Web Authentication` database method is in charge of managing web server engine access. It is called by 4D or 4D Server when a dynamic HTTP request is received.

### Database method calls

The `On Web Authentication` database method is automatically called when a request or processing requires the execution of some 4D code (except for REST calls). It is also called when the web server receives an invalid static URL (for example, if the

static page requested does not exist).

The `On Web Authentication` database method is therefore called:

- when the web server receives a URL requesting a resource that does not exist
- when the web server receives a URL beginning with `4DACTION/`, `4DCGI/...`
- when the web server receives a root access URL and no home page has been set in the Settings or by means of the `WEB SET HOME PAGE` command
- when the web server processes a tag executing code (e.g `4DSCRIPT`) in a semi-dynamic page.

The `On Web Authentication` database method is NOT called:

- when the web server receives a URL requesting a valid static page.
- when the web server receives a URL beginning with `rest/` and the REST server is launched (in this case, the authentication is handled through the [On REST Authentication database method](#) or [Structure settings](#)).

## Syntax

**On Web Authentication**( \$1 : Text ; \$2 : Text ; \$3 : Text ; \$4 : Text ; \$5 : Text ; \$6 : Text ) -> \$0 : Boolean

Parameters	Type	Description
\$1	Text	<-URL
\$2	Text	<-HTTP headers + HTTP body (up to 32 kb limit)
\$3	Text	<-IP address of the web client (browser)
\$4	Text	<-IP address of the server
\$5	Text	<-User name
\$6	Text	<-Password
\$0	Boolean	-> True = request accepted, False = request rejected

You must declare these parameters as follows:

```
//On Web Authentication database method  
  
C_TEXT($1;$2;$3;$4;$5;$6)  
C_BOOLEAN($0)  
  
//Code for the method
```

Alternatively, you can use the [named parameters](#) syntax:

```
// On Web Authentication database method  
#DECLARE ($url : Text; $header : Text; \  
$BrowserIP : Text; $ServerIP : Text; \  
$user : Text; $password : Text) \  
-> $RequestAccepted : Boolean
```

All the `On Web Authentication` database method's parameters are not necessarily filled in. The information received by the database method depends on the selected [authentication mode](#)).

### \$1 - URL

The first parameter (`$1`) is the URL received by the server, from which the host address has been removed.

Let's take the example of an Intranet connection. Suppose that the IP address of your 4D Web Server machine is 123.45.67.89. The following table shows the values of \$1 depending on the URL entered in the Web browser:

URL entered in web browser	Value of parameter \$1
123.45.67.89	/
<a href="http://123.45.67.89">http://123.45.67.89</a>	/
123.45.67.89/Customers	/Customers
<a href="http://123.45.67.89/Customers/Add">http://123.45.67.89/Customers/Add</a>	/Customers/Add
123.45.67.89/Do_This/If_OK/Do_That/Do_This/If_OK/Do_That	

## \$2 - Header and Body of the HTTP request

The second parameter (\$2) is the header and the body of the HTTP request sent by the web browser. Note that this information is passed to your `On Web Authentication` database method as it is. Its contents will vary depending on the nature of the web browser which is attempting the connection.

If your application uses this information, it is up to you to parse the header and the body. You can use the `WEB GET HTTP HEADER` and the `WEB GET HTTP BODY` commands.

For performance reasons, the size of data passing through the \$2 parameter must not exceed 32 KB. Beyond this size, they are truncated by the 4D HTTP server.

## \$3 - Web client IP address

The \$3 parameter receives the IP address of the browser's machine. This information can allow you to distinguish between intranet and internet connections.

4D returns IPv4 addresses in a hybrid IPv6/IPv4 format written with a 96-bit prefix, for example ::ffff:192.168.2.34 for the IPv4 address 192.168.2.34. For more information, refer to the [IPv6 Support](#) section.

## \$4 - Server IP address

The \$4 parameter receives the IP address used to call the web server. 4D allows for multi-homing, which allows you to exploit machines with more than one IP address. For more information, please refer to the [Configuration page](#).

## \$5 and \$6 - User Name and Password

The \$5 and \$6 parameters receive the user name and password entered by the user in the standard identification dialog box displayed by the browser. This dialog box appears for each connection, if `basic` or `digest` authentication is selected.

If the user name sent by the browser exists in 4D, the \$6 parameter (the user's password) is not returned for security reasons.

## \$0 parameter

The `On Web Authentication` database method returns a boolean in \$0:

- If \$0 is True, the connection is accepted.
- If \$0 is False, the connection is refused.

The `On Web Connection` database method is only executed if the connection has been accepted by `On Web Authentication`.

### **WARNING**

If no value is set to \$0 or if \$0 is not defined in the `On Web Authentication` database method, the connection is considered as accepted and the `On Web Connection` database method is executed.

- Do not call any interface elements in the `On Web Authentication` database method (`ALERT`, `DIALOG`, etc.) because otherwise its execution will be interrupted and the connection refused. The same thing will happen if an error occurs during its processing.

### **Example**

Example of the `On Web Authentication` database method in [DIGEST mode](#):

```
// On Web Authentication Database Method
#DECLARE ($url : Text; $header : Text; $ipB : Text; $ipS : Text; \
$user : Text; $pw : Text) -> $valid : Boolean

var $found : cs.WebUserSelection
$valid:=False

$found:=ds.WebUser.query("User === :1";$user)
If($found.length=1) // User is found
    $valid:=WEB Validate digest($user; [WebUser]password)
Else
    $valid:=False // User does not exist
End if
```

## User sessions

The 4D web server provides built-in features for managing **user sessions**. Creating and maintaining user sessions allows you to control and improve the user experience on your web application. When user sessions are enabled, web clients can reuse the same server context from one request to another.

Web server user sessions allow to:

- handle multiple requests simultaneously from the same web client through an unlimited number of preemptive processes (web server sessions are **scalable**),
- share data between the processes of a web client,
- associate privileges to user sessions,
- handle access through a `Session` object and the [Session API](#).

**Note:** The current implementation is only the first step of an upcoming comprehensive feature allowing developers to manage hierarchical user permissions through sessions in the whole web application.

## Enabling sessions

The session management feature can be enabled and disabled on your 4D web server. There are different ways to enable session management:

- Using the **Scalable sessions** option on the "Web/Options (I)" page of the Settings (permanent setting):

This option is selected by default in new projects. It can however be disabled by selecting the **No sessions** option, in which case the web session features are disabled (no `Session` object is available).

- Using the `.scalableSession` property of the Web Server object (to pass in the `settings` parameter of the `.start()` function). In this case, this setting overrides the option defined in the Settings dialog box for the Web Server object (it is not stored on disk).

The `WEB SET OPTION` command can also set the session mode for the main Web server.

In any cases, the setting is local to the machine; so it can be different on the 4D Server Web server and the Web servers of remote 4D machines.

**Compatibility:** A **Legacy sessions** option is available in projects created with a 4D version prior to 4D v18 R6 (for more information, please refer to the [doc.4d.com](#) web site).

## Session implementation

When [sessions are enabled](#), automatic mechanisms are implemented, based upon a private cookie set by 4D itself: "4DSID\_AppName", where *AppName* is the name of the application project. This cookie references the current web session for the application.

The cookie name can be get using the `.sessionCookieName` property.

1. In each web client request, the Web server checks for the presence and the value of the private "4DSID\_AppName" cookie.
2. If the cookie has a value, 4D looks for the session that created this cookie among the existing sessions; if this session is found, it is reused for the call.
3. If the client request does not correspond to an already opened session:
  - a new session with a private "4DSID\_AppName" cookie is created on the web server
  - a new Guest `Session` object is created and is dedicated to the scalable web session.

The current `Session` object can then be accessed through the `Session` command in the code of any web processes.

Web processes usually do not end, they are recycled in a pool for efficiency. When a process finishes executing a request, it is put back in the pool and made available for the next request. Since a web process can be reused by any session, [process variables](#) must be cleared by your code at the end of its execution (using [CLEAR VARIABLE](#) for example). This cleanup is necessary for any process related information, such as a reference to an opened file. This is the reason why **it is recommended** to use the [Session](#) object when you want to keep session related information.

## Preemptive mode

On 4D Server, Web server sessions are automatically handled through preemptive processes, **even in interpreted mode**. You need to make sure that your web server code is [compliant with a preemptive execution](#).

To debug interpreted web code on the server machine, make sure the debugger is [attached to the server](#) or [to a remote machine](#). Web processes then switch to cooperative mode and the web server code can be debugged.

With 4D single-user, interpreted code always runs in cooperative mode.

## Sharing information

Each `Session` object provides a [.storage](#) property which is a [shared object](#). This property allows you to share information between all processes handled by the session.

## Session lifetime

A scalable web session is closed when:

- the web server is stopped,
- the timeout of the session cookie has been reached.

The lifespan of an inactive cookie is 60 minutes by default, which means that the web server will automatically close inactive sessions after 60 minutes.

This timeout can be set using the [.idleTimeout](#) property of the `Session` object (the timeout cannot be less than 60 minutes).

When a scalable web session is closed, if the [Session](#) command is called afterwards:

- the `Session` object does not contain privileges (it is a Guest session)
- the `.storage` property is empty
- a new session cookie is associated to the session

## Privileges

Privileges can be associated to sessions. On the web server, you can provide specific access or features depending on the privileges of the session.

You can assign privileges using the `.setPrivileges()` function. In your code, you can check the session's privileges to allow or deny access using the `.hasPrivilege()` function. By default, new sessions do not have any privilege: they are **guest** sessions (`.isGuest()` function returns true).

In the current implementation (v18 R6), only the "WebAdmin" privilege is available.

Example:

```
If (Session.hasPrivilege("WebAdmin"))
    //Access is granted, do nothing
Else
    //Display an authentication page
End if
```

## Example

In a CRM application, each salesperson manages their own client portfolio. The datastore contains at least two linked dataclasses: Customers and SalesPersons (a salesperson has several customers).

We want a salesperson to authenticate, open a session on the web server, and have the top 3 customers be loaded in the session.

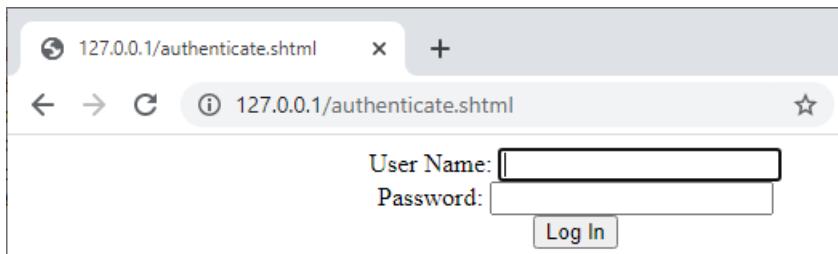
1. We run this URL to open a session:

`http://localhost:8044/authenticate.shtml`

In a production environment, it is necessary to use a [HTTPS connection](#) to avoid any uncrypted information to circulate on the network.

2. The `authenticate.shtml` page is a form containing `userId` et `password` input fields and sending a 4ACTION POST action:

```
<!DOCTYPE html>
<html>
<body bgcolor="#ffffff">
<FORM ACTION="/4ACTION/authenticate" METHOD=POST>
    UserId: <INPUT TYPE=TEXT NAME=userId VALUE=""><br/>
    Password: <INPUT TYPE=TEXT NAME=password VALUE=""><br/>
<INPUT TYPE=SUBMIT NAME=OK VALUE="Log In">
</FORM>
</body>
</html>
```



3. The authenticate project method looks for the *userID* person and validates the password against the hashed value already stored in the *SalesPersons* table:

```
var $indexUserId; $indexPassword; $userId : Integer
var $password : Text
var $userTop3; $sales; $info : Object

ARRAY TEXT($anames; 0)
ARRAY TEXT($avalues; 0)

WEB GET VARIABLES($anames; $avalues)

$indexUserId:=Find in array($anames; "userId")
$userId:=Num($avalues{$indexUserId})

$indexPassword:=Find in array($anames; "password")
$password:=$avalues{$indexPassword}

$sales:=ds.SalesPersons.query("userId = :1"; $userId).first()

If ($sales#Null)
    If (Verify password hash($password; $sales.password) )
        $info:=New object()
        $info.userName:=$sales.firstname+" "+$sales.lastname
        Session.setPrivileges($info)
        Use (Session.storage)
            If (Session.storage.myTop3=NULL)
                $userTop3:=$sales.customers.orderBy("totalPurchase
desc").slice(0; 3)
                Session.storage.myTop3:=$userTop3
            End if
        End use
        WEB SEND HTTP REDIRECT("/authenticationOK.shtml")
    Else
        WEB SEND TEXT("This password is wrong")
    End if
Else
    WEB SEND TEXT("This userId is unknown")
End if
```

## Using preemptive web processes

The 4D Web Server allows you to take full advantage of multi-core computers by using preemptive web processes in your applications. You can configure your web-related code, including 4D tags, web database methods or ORDA REST class functions to run simultaneously on as many cores as possible.

For in-depth information on preemptive process in 4D, please refer to the *Preemptive 4D processes* section in the [4D Language Reference](#).

## Availability of preemptive mode for web processes

The following table indicates whether the preemptive mode is used or is available, depending on the execution context:

4D Server	Interpreted ( <a href="#">debugger attached</a> )	Interpreted (debugger detached)	Compiled
REST Server	cooperative	preemptive	preemptive
Web Server	cooperative	<i>web setting</i>	<i>web setting</i>
Web Services Server	cooperative	<i>web setting</i>	<i>web setting</i>
<b>4D remote/single-user</b> Interpreted Compiled			
REST Server	cooperative	preemptive	
Web Server	cooperative	<i>web setting</i>	
Web Services Server	cooperative	<i>web setting</i>	

- REST Server: handles [ORDA data model class functions](#)
- Web Server: handles [web templates](#), [4DACTION](#) and [database methods](#)
- Web Service Server: handles SOAP requests
- ***web setting*** means that the preemptive mode depends on a setting value:
  - when [Scalable sessions](#) option is selected, the [preemptive mode is automatically used](#) for web processes.
  - otherwise, the [Use preemptive processes](#) option is taken into account.
  - regarding Web service processes (server or client), preemptive mode is supported at method level. You just have to select "Can be run in preemptive processes" property for published SOAP server methods (see [Publishing a Web Service with 4D](#)) or proxy client methods (see [Subscribing to a Web Service in 4D](#)) and make sure they are confirmed thread-safe by the compiler.

## Writing thread-safe web server code

All 4D code executed by the web server must be thread-safe if you want your web processes to be run in preemptive mode. When the [preemptive mode is enabled](#), the following parts of the application will be automatically evaluated by the 4D compiler:

- All web-related database methods:
  - [On Web Authentication](#)
  - [On Web Connection](#)
  - [On REST Authentication](#)
  - [On Mobile App Authentication](#) and [On Mobile App Action](#)

- The `compiler_web` project method (regardless of its actual "Execution mode" property);
- Basically any code processed by the `PROCESS 4D TAGS` command in the web context, for example through .shtml pages
- Any project method with the "Available through 4D tags and URLs (`^DACTION`, etc.)" attribute
- Triggers for tables with "Expose as REST resource" attribute
- [ORDA data model class functions](#) called via REST

For each of these methods and code parts, the compiler will check if the thread-safety rules are respected, and will return errors in case of issues. For more information about thread-safety rules, please refer to the *Writing a thread-safe method* paragraph in the *Processes* chapter of the [4D Language Reference](#) manual.

## Thread-safety of 4D web code

Most of the web-related 4D commands and functions, database methods and URLs are thread-safe and can be used in preemptive mode.

### 4D commands and database methods

All 4D web-related commands are thread-safe, *i.e.*:

- all commands from the *Web Server* theme,
- all commands from the *HTTP Client* theme.

The web-related database methods are thread-safe and can be used in preemptive mode (see above): [On Web Authentication](#), [On Web Connection](#), [On REST Authentication...](#).

Of course, the code executed by these methods must also be thread-safe.

### Web Server URLs

The following 4D Web Server URLs are thread-safe and can be used in preemptive mode:

- `4daction/` (the called project method must also be thread-safe)
- `4dcgi/` (the called database methods must also be thread-safe)
- `4dwebtest/`
- `4dblank/`
- `4dstats/`
- `4dhtmlstats/`
- `4dcache-clear/`
- `rest/`
- `4dim-field/` (generated by `PROCESS 4D TAGS` for web request on picture fields)
- `4dim/` (generated by `PROCESS 4D TAGS` for web request on picture variables)

### Preemptive web process icon

Both the Runtime Explorer and the 4D Server administration window display a specific icon for preemptive web processes:

Process type	Icon
Preemptive web method	

# REST Server

Configuring the 4D REST Server and using the REST API.

## Guides

[6 items](#)

## API (general)

[4 items](#)

## API (dataClass)

[22 items](#)

# Getting Started

4D provides you with a powerful REST server, that allows direct access to data stored in your 4D applications.

The REST server is included in 4D and 4D Server, it is automatically available in your 4D applications [once it is configured](#).

This section is intended to help familiarize you with REST functionality by means of a simple example. We are going to:

- create and configure a basic 4D application project
- access data from the 4D project through REST using a standard browser.

To keep the example simple, we're going to use 4D and a browser that are running on the same machine. Of course, you could also use a remote architecture.

## Creating and configuring the 4D project

1. Launch your 4D or 4D Server application and create a new project. You can name it "Emp4D", for example.
2. In the Structure editor, create an [Employees] table and add the following fields to it:
  - Lastname (Alpha)
  - Firstname (Alpha)
  - Salary (Longint)

Employees	
ID	2 <sup>32</sup>
Lastname	A
Firstname	A
Salary	2 <sup>32</sup>

The "Expose a REST resource" option is checked by default for the table and every field; do not change this setting.

3. Create forms, then create a few employees:

Emp4D - Employees: 3 of 3			
ID :	Lastname :	Firstname :	Salary :
1	Brown	Michael	25000
2	Jones	Maryanne	35000
3	Smithers	Jack	41000

4. Open the **Web > Web Features** page of the Settings dialog box and [check the Expose as REST server](#) option.
5. In the **Run** menu, select **Start Web Server** (if necessary), then select **Test Web Server**.

4D displays the default home page of the 4D Web Server.

## Accessing 4D data through the browser

You can now read and edit data within 4D only through REST requests.

Any 4D REST URL request starts with `/rest`, to be inserted after the `address:port` area. For example, to see what's inside the 4D datastore, you can write:

```
http://127.0.0.1/rest/$catalog
```

The REST server replies:

```
{
  "__UNIQID": "96A49F7EF2ABDE44BF32059D9ABC65C1",
  "dataClasses": [
    {
      "name": "Employees",
      "uri": "/rest/$catalog/Employees",
      "dataURI": "/rest/Employees"
    }
  ]
}
```

It means that the datastore contains the Employees dataclass. You can see the dataclass attributes by typing:

```
/rest/$catalog/Employees
```

If you want to get all entities of the Employee dataclass, you write:

```
/rest/Employees
```

**Response:**

```
{
    "__entityModel": "Employees",
    "__GlobalStamp": 0,
    "__COUNT": 3,
    "__FIRST": 0,
    "__ENTITIES": [
        {
            "__KEY": "1",
            "__TIMESTAMP": "2020-01-07T17:07:52.467Z",
            "__STAMP": 2,
            "ID": 1,
            "Lastname": "Brown",
            "Firstname": "Michael",
            "Salary": 25000
        },
        {
            "__KEY": "2",
            "__TIMESTAMP": "2020-01-07T17:08:14.387Z",
            "__STAMP": 2,
            "ID": 2,
            "Lastname": "Jones",
            "Firstname": "Maryanne",
            "Salary": 35000
        },
        {
            "__KEY": "3",
            "__TIMESTAMP": "2020-01-07T17:08:34.844Z",
            "__STAMP": 2,
            "ID": 3,
            "Lastname": "Smithers",
            "Firstname": "Jack",
            "Salary": 41000
        }
    ],
    "__SENT": 3
}
```

You have many possibilities to filter data to receive. For example, to get only the "Lastname" attribute value from the 2nd entity, you can just write:

```
/rest/Employees(2)/Lastname
```

### **Response:**

```
{
    "__entityModel": "Employees",
    "__KEY": "2",
    "__TIMESTAMP": "2020-01-07T17:08:14.387Z",
    "__STAMP": 2,
    "Lastname": "Jones"
}
```

The 4D [REST API](#) provides various commands to interact with the 4D applications.

## Server Configuration

Using standard HTTP requests, the 4D REST Server allows external applications to access the data of your application directly, *i.e.* to retrieve information about the dataclasses in your project, manipulate data, log into your web application, and much more.

To start using the REST features, you need to start and configure the 4D REST server.

- On 4D Server, opening a REST session requires that a free 4D client licence is available.
- On 4D single-user, you can open up to three REST sessions for testing purposes.
- You need to manage the [session](#) for your requesting application.

## Starting the REST Server

For security reasons, by default, 4D does not respond to REST requests. If you want to start the REST Server, you must check the **Expose as REST server** option in the **Web > Web Features** page of the structure settings in order for REST requests to be processed.

REST services use the 4D HTTP server, so you need to make sure that the 4D web server is started.

The warning message "Caution, check the access privileges" is displayed when you check this option to draw your attention to the fact that when REST services are activated, by default access to database objects is free as long as the REST accesses have not been configured.

You must restart the 4D application for your changes to take effect.

## Configuring REST access

By default, REST accesses are open to all users which is obviously not recommended for security reasons, and also to control client licenses usage.

You can configuring REST accesses with one of the following means:

- assigning a **Read/Write** user group to REST services in the "**Web > Web Features**" page of the Structure Settings;
- writing an `On REST Authentication` database method to intercept and handle every initial REST request.

You cannot use both features simultaneously. Once an `On REST Authentication` database method has been defined, 4D fully delegates control of REST requests to it: any setting made using the "Read/Write" menu on the **Web > Web Features** page of the Structure Settings is ignored.

## Using the Structure Settings

The **Read/Write** menu in the "**Web > Web Features**" page of the structure settings specifies a group of 4D users that is authorized to establish the link to the 4D application using REST queries.

By default, the menu displays \<Anyone>, which means that REST accesses are open to all users. Once you have specified a group, only a 4D user account that belongs to this group may be used to [access 4D by means of a REST request](#). If an account is used that does not belong to this group, 4D returns an authentication error to the sender of the request.

In order for this setting to take effect, the `On REST Authentication` database method must not be defined. If it exists, 4D ignores access settings defined in the Structure Settings.

## Using the On REST Authentication database method

The `On REST Authentication` database method provides you with a custom way of controlling the opening of REST sessions on 4D. This database method is automatically called when a new session is opened through a REST request. When a [request to open a REST session](#) is received, the connection identifiers are provided in the header of the request. The `On REST Authentication` database method is called so that you can evaluate these identifiers. You can use the list of users for the 4D application or you can use your own table of identifiers. For more information, refer to the `On REST Authentication` database method [documentation](#).

## Exposing tables and fields

Once REST services are enabled in the 4D application, by default a REST session can access all tables and fields of the 4D database through the [datastore interface](#). Thus, it can use their data. For example, if your database contains an [Employee] table, it is possible to write:

```
http://127.0.0.1:8044/rest/Employee/?$filter="salary>10000"
```

This request will return all employees whose salary field is higher than 10000.

4D tables and/or fields that have the "Invisible" attribute are also exposed in REST by default.

If you want to customize the datastore objects accessible through REST, you must disable the exposure of each table and/or field that you want to hide. When a REST request attempts to access an unauthorized resource, 4D returns an error.

### Exposing tables

By default, all tables are exposed in REST.

For security reasons, you may want to only expose certain tables of your datastore to REST calls. For instance, if you created a [Users] table storing user names and passwords, it would be better not to expose it.

To remove the REST exposure for a table:

1. Display the Table Inspector in the Structure editor and select the table you want to modify.
2. Uncheck the **Expose as REST resource** option:  
Do this for each table whose exposure needs to be modified.

### Exposing fields

By default, all 4D database fields are exposed in REST.

You may not want to expose certain fields of your tables to REST. For example, you may not want to expose the [Employees]Salary field.

To remove the REST exposure for a field:

1. Display the Field Inspector in the Structure editor and select the field you want to modify.

2. Uncheck the **Expose as REST resource** for the field.

Repeat this for each field whose exposure needs to be modified.

In order for a field to be accessible through REST, the parent table must be as well. If the parent table is not exposed, none of its fields will be, regardless of their status.

## Users and sessions

REST requests can benefit from [web user sessions](#), providing extra features such as multiple requests handling, data sharing between the web client processes, and user privileges.

As a first step to open a REST session on the 4D server, the user sending the request must be authenticated.

## Authenticating users

You log in a user to your application by calling [`\$directory/login`](#) in a POST request including the user's name and password in the header. This request calls the `On REST Authentication` database method (if it exists), where you can check the user's credentials (see example below).

## Opening sessions

When [scalable sessions are enabled](#) (recommended), if the `On REST Authentication` database method returns `true`, a user session is then automatically opened and you can handle it through the `Session` object and the [Session API](#). Subsequent REST requests will reuse the same session cookie.

If the `On REST Authentication` database method has not been defined, a `guest` session is opened.

## Example

In this example, the user enters their email and password in an html page that requests [`\$directory/login`](#) in a POST (it is recommended to use an HTTPS connection to send the html page). The `On REST Authentication` database method is called to validate the credentials and to set the session.

The HTML login page:

The image shows a simple HTML form for logging in. It consists of three elements: a text input field labeled "Email:", a text input field labeled "Password:", and a blue "Login" button. All elements are contained within a single vertical column.

Email:	<input type="text"/>
Password:	<input type="password"/>
<input type="button" value="Login"/>	

```

<html><body bgcolor="#ffffff">

<div id="demo">
    <FORM name="myForm">
Email: <INPUT TYPE=TEXT NAME=userId VALUE=""><br/>
Password: <INPUT TYPE=TEXT NAME=password VALUE=""><br/>
<button type="button" onclick="onClick()">
Login
</button>
<div id="authenticationFailed"
style="visibility:hidden;">Authentication failed</div>
</FORM>
</div>

<script>
function sendData(data) {
    var XHR = new XMLHttpRequest();

    XHR.onreadystatechange = function() {
        if (this.status == 200) {
            window.location = "authenticationOK.shtml";
        }
        else {
            document.getElementById("authenticationFailed").style.visibility =
= "visible";
        }
    };

    XHR.open('POST', 'http://127.0.0.1:8044/rest/$directory/login');
//rest server address

    XHR.setRequestHeader('username-4D', data.userId);
    XHR.setRequestHeader('password-4D', data.password);
    XHR.setRequestHeader('session-4D-length', data.timeout);

    XHR.send();
};

function onClick()
{
sendData({userId:document.forms['myForm'].elements['userId'].value ,
password:document.forms['myForm'].elements['password'].value ,
timeout:120});
}
</script></body></html>

```

When the login page is sent to the server, the `On REST Authentication` database method is called:

```

//On REST Authentication

#DECLARE($userId : Text; $password : Text) -> $Accepted : Boolean
var $sales : cs.SalesPersonsEntity

$Accepted:=False

    //A '/rest' URL has been called with headers username-4D and
    password-4D
If ($userId#"")
    $sales:=ds.SalesPersons.query("email = :1"; $userId).first()
    If ($sales#Null)
        If (Verify password hash($password; $sales.password))
            fillSession($sales)
            $Accepted:=True
        End if
    End if
End if

```

As soon as it has been called and returned `True`, the `On REST Authentication` database method is no longer called in the session.

The `fillSession` project method initializes the user session, for example:

```

#DECLARE($sales : cs.SalesPersonsEntity)
var $info : Object

$info:=New object()
$info.userName:=$sales.firstname+" "+$sales.lastname

Session.setPrivileges($info)

Use (Session.storage)
    If (Session.storage.myTop3=Null)

Session.storage.myTop3:=$sales.customers.orderBy("totalPurchase
desc").slice(0; 3)
    End if
End use

```

# Getting Server Information

You can get several information from the REST server:

- the exposed datastores and their attributes
- the REST server cache contents, including user sessions.

## Catalog

Use the `$catalog`, `$catalog/{dataClass}`, or `$catalog/$all` parameters to get the list of [exposed dataclasses and their attributes](#).

To get the collection of all exposed dataclasses along with their attributes:

```
GET /rest/$catalog/$all
```

## Cache info

Use the `$info` parameter to get information about the entity selections currently stored in 4D Server's cache as well as running user sessions.

## queryPath and queryPlan

Entity selections that are generated through queries can have the following two properties: `queryPlan` and `queryPath`. To calculate and return these properties, you just need to add `$queryPlan` and/or `$queryPath` in the REST request.

For example:

```
GET /rest/People/$filter="employer.name=acme AND  
lastName=Jones"&$queryplan=true&$querypath=true
```

These properties are objects that contain information about how the server performs composite queries internally through dataclasses and relations:

- **queryPlan**: object containing the detailed description of the query just before it was executed (i.e., the planned query).
- **queryPath**: object containing the detailed description of the query as it was actually performed.

The information recorded includes the query type (indexed and sequential) and each necessary subquery along with conjunction operators. Query paths also contain the number of entities found and the time required to execute each search criterion. You may find it useful to analyze this information while developing your application. Generally, the description of the query plan and its path are identical but they can differ because 4D can implement dynamic optimizations when a query is executed in order to improve performance. For example, the 4D engine can dynamically convert an indexed query into a sequential one if it estimates that it is faster. This particular case can occur when the number of entities being searched for is low.

# Manipulating Data

All [exposed dataclasses, attributes](#) and [functions](#) can be accessed through REST. Dataclass, attribute, and function names are case-sensitive; however, the data for queries is not.

## Querying data

To query data directly, you can do so using the [\\$filter](#) function. For example, to find a person named "Smith", you could write:

```
http://127.0.0.1:8081/rest/Person/?$filter="lastName=Smith"
```

## Adding, modifying, and deleting entities

With the REST API, you can perform all the manipulations to data as you can in 4D.

To add and modify entities, you can call [\\$method=update](#). If you want to delete one or more entities, you can use [\\$method=delete](#).

Besides retrieving a single entity in a dataclass using [{dataClass}\({key}\)](#), you can also write a [class function](#) that returns an entity selection (or a collection).

Before returning a selection, you can also sort it by using [\\$orderby](#) one or more attributes (even relation attributes).

## Navigating data

Add the [\\$skip](#) (to define with which entity to start) and [\\$stop/\\$limit](#) (to define how many entities to return) REST requests to your queries or entity selections to navigate the collection of entities.

## Creating and managing entity set

An entity set (aka *entity selection*) is a collection of entities obtained through a REST request that is stored in 4D Server's cache. Using an entity set prevents you from continually querying your application for the same results. Accessing an entity set is much quicker and can improve the speed of your application.

To create an entity set, call [\\$method=entityset](#) in your REST request. As a measure of security, you can also use [\\$savedfilter](#) and/or [\\$savedorderby](#) when you call [\\$filter](#) and/or [\\$orderby](#) so that if ever the entity set timed out or was removed from the server, it can be quickly retrieved with the same ID as before.

To access the entity set, you must use [\\$entityset/{entitySetID}](#), for example:

```
/rest/People/$entityset/0AF4679A5C394746BFEB68D2162A19FF
```

By default, an entity set is stored for two hours; however, you can change the timeout by passing a new value to [\\$timeout](#). The timeout is continually being reset to the value defined for its timeout (either the default one or the one you define) each time you use it.

If you want to remove an entity set from 4D Server's cache, you can use [\\$method=release](#).

If you modify any of the entity's attributes in the entity set, the values will be updated. However, if you modify a value that was a part of the query executed to create the entity set, it will not be removed from the entity set even if it no longer fits the search criteria. Any entities you delete will, of course, no longer be a part of the entity set.

If the entity set no longer exists in 4D Server's cache, it will be recreated with a new default timeout of 10 minutes. The entity set will be refreshed (certain entities might be included while others might be removed) since the last time it was created, if it no longer existed before recreating it.

Using `$entityset/{entitySetID}?$logicOperator... &$otherCollection`, you can combine two entity sets that you previously created. You can either combine the results in both, return only what is common between the two, or return what is not common between the two.

A new selection of entities is returned; however, you can also create a new entity set by calling `$method=entityset` at the end of the REST request.

## Calculating data

By using `$compute`, you can compute the **average**, **count**, **min**, **max**, or **sum** for a specific attribute in a dataclass. You can also compute all values with the `$all` keyword.

For example, to get the highest salary:

```
/rest/Employee/salary/?$compute=max
```

To compute all values and return a JSON object:

```
/rest/Employee/salary/?$compute=$all
```

## Calling Data model class functions

You can call ORDA Data Model [user class functions](#) through POST requests, so that you can benefit from the exposed API of the targeted application. For example, if you have defined a `getCity()` function in the City dataclass class, you could call it using the following request:

```
/rest/City/getCity
```

with data in the body of the request: `["Paris"]`

Calls to 4D project methods that are exposed as REST Service are still supported but are deprecated.

## Selecting Attributes to get

You can always define which attributes to return in the REST response after an initial request by passing their path in the request (e.g., `Company(1)/name, revenues/`)

You can apply this filter in the following ways:

Object	Syntax
Dataclass {dataClass}/{att1,att2...}	/People/firstName,lastName
Collection of entities	{dataClass}/{att1,att2...}/?\$filter="{filter}" /People/firstName,lastName
Specific entity	{dataClass}({ID})/{att1,att2...} /People(1)/firstName,lastName
Entity selection	{dataClass}/{att1,att2...}/\$entityset/{entitySetID}/People/firstName/\$entity /People:firstName(Larry),

The attributes must be delimited by a comma, *i.e.*,

`/Employee/firstName,lastName,salary`. Storage or relation attributes can be passed.

## Examples

Here are a few examples, showing you how to specify which attributes to return depending on the technique used to retrieve entities.

You can apply this technique to:

- Dataclasses (all or a collection of entities in a dataclass)
- Specific entities
- Entity sets

### Dataclass Example

The following requests returns only the first name and last name from the People dataclass (either the entire dataclass or a selection of entities based on the search defined in `$filter`).

`GET /rest/People/firstName,lastName/`

### Result:

```
{
    __entityModel: "People",
    __COUNT: 4,
    __SENT: 4,
    __FIRST: 0,
    __ENTITIES: [
        {
            __KEY: "1",
            __STAMP: 1,
            firstName: "John",
            lastName: "Smith"
        },
        {
            __KEY: "2",
            __STAMP: 2,
            firstName: "Susan",
            lastName: "O'Leary"
        },
        {
            __KEY: "3",
            __STAMP: 2,
            firstName: "Pete",
            lastName: "Marley"
        },
        {
            __KEY: "4",
            __STAMP: 1,
            firstName: "Beth",
            lastName: "Adams"
        }
    ]
}
```

```
GET /rest/People/firstName,lastName/?$filter="lastName='A@'" /
```

### **Result:**

```
{
    __entityModel: "People",
    __COUNT: 1,
    __SENT: 1,
    __FIRST: 0,
    __ENTITIES: [
        {
            __KEY: "4",
            __STAMP: 4,
            firstName: "Beth",
            lastName: "Adams"
        }
    ]
}
```

### **Entity Example**

The following request returns only the first name and last name attributes from a specific entity in the People dataclass:

```
GET /rest/People(3)/firstName,lastName/
```

### **Result:**

```
{  
    __entityModel: "People",  
    __KEY: "3",  
    __STAMP: 2,  
    firstName: "Pete",  
    lastName: "Marley"  
}
```

```
GET /rest/People(3) /
```

## Result:

```
{  
    __entityModel: "People",  
    __KEY: "3",  
    __STAMP: 2,  
    ID: 3,  
    firstName: "Pete",  
    lastName: "Marley",  
    salary: 30000,  
    employer: {  
        __deferred: {  
            uri: "http://127.0.0.1:8081/rest/Company(3)",  
            __KEY: "3"  
        }  
    },  
    fullName: "Pete Marley",  
    employerName: "microsoft"  
}
```

## Entity Set Example

Once you have [created an entity set](#), you can filter the information in it by defining which attributes to return:

```
GET  
/rest/People/firstName,employer.name/$entityset/BDCD8AABE13144118A4CF8  
641D5883F5?$expand=employer
```

## Viewing an image attribute

If you want to view an image attribute in its entirety, write the following:

```
GET /rest/Employee(1)/photo?$imageformat=best&$version=1&$expand=photo
```

For more information about the image formats, refer to [\\$imageformat](#). For more information about the version parameter, refer to [\\$version](#).

## Saving a BLOB attribute to disk

If you want to save a BLOB stored in your dataclass, you can write the following:

```
GET /rest/Company(11)/blobAtt?$binary=true&$expand=blobAtt
```

## Retrieving only one entity

You can use the [{dataClass}:{attribute}\(value\)](#) syntax when you want to retrieve only one entity. It's especially useful when you want to do a related search that isn't created on the dataclass's primary key. For example, you can write:

```
GET /rest/Company:companyCode ("Acme001")
```

## Calling ORDA class functions

You can call [data model class functions](#) defined for the ORDA Data Model through your REST requests, so that you can benefit from the exposed API of the targeted 4D application.

Functions are simply called in POST requests on the appropriate ORDA interface, without (). For example, if you have defined a `getCity()` function in the City dataclass class, you could call it using the following request:

```
/rest/City/getCity
```

with data in the body of the POST request: `[ "Aguada" ]`

In 4D language, this call is equivalent to :

```
$city:=ds.City.getCity("Aguada")
```

Only functions with the `exposed` keyword can be directly called from REST requests. See [Exposed vs non-exposed functions](#) section.

## Function calls

Functions must always be called using REST **POST** requests (a GET request will receive an error).

Functions are called on the corresponding object on the server datastore.

Class function	Syntax
<a href="#">datastore class</a>	<code>/rest/\$catalog/DataStoreClassFunction</code>
<a href="#">dataclass class</a>	<code>/rest/{dataClass}/DataClassClassFunction</code>
<a href="#">entitySelection class</a>	<code>/rest/{dataClass}/EntitySelectionClassFunction</code> <code>/rest/{dataClass}/EntitySelectionClassFunction/\$entityset/e</code> <code>/rest/{dataClass}/EntitySelectionClassFunction/\$filter</code> <code>/rest/{dataClass}/EntitySelectionClassFunction/\$orderby</code>
<a href="#">entity class</a>	<code>/rest/{dataClass} (key)/EntityClassFunction/</code>

`/rest/{dataClass}/Function` can be used to call either a dataclass or an entity selection function (`/rest/{dataClass}` returns all entities of the DataClass as an entity selection).

The function is searched in the entity selection class first. If not found, it is searched in the dataclass. In other words, if a function with the same name is defined in both the DataClass class and the EntitySelection class, the dataclass class function will never be executed.

All 4D code called from REST requests **must be thread-safe** if the project runs in compiled mode, because the REST Server always uses preemptive processes in this case (the [Use preemptive process setting value](#) is ignored by the REST Server).

## Parameters

You can send parameters to functions defined in ORDA user classes. On the server side, they will be received in the class functions in regular \$1, \$2, etc. parameters.

The following rules apply:

- Parameters must be passed in the **body of the POST request**
- Parameters must be enclosed within a collection (JSON format)
- All scalar data types supported in JSON collections can be passed as parameters.
- Entity and entity selection can be passed as parameters. The JSON object must contain specific attributes used by the REST server to assign data to the corresponding ORDA objects: **DATACLASS**, **ENTITY**, **ENTITIES**, **DATASET**.

See [this example](#) and [this example](#).

## Scalar value parameter

Parameter(s) must simply be enclosed in a collection defined in the body. For example, with a dataclass function `getCities()` receiving text parameters:

```
/rest/City/getCities
```

**Parameters in body:** ["Aguada","Paris"]

All JSON data types are supported in parameters, including JSON pointers. Dates can be passed as strings in ISO 8601 date format (e.g. "2020-08-22T22:00:00Z").

## Entity parameter

Entities passed in parameters are referenced on the server through their key (*i.e.* `__KEY` property). If the key parameter is omitted in a request, a new entity is loaded in memory the server. You can also pass values for any attributes of the entity. These values will automatically be used for the entity handled on the server.

If the request sends modified attribute values for an existing entity on the server, the called ORDA data model function will be automatically executed on the server with modified values. This feature allows you, for example, to check the result of an operation on an entity, after applying all business rules, from the client application. You can then decide to save or not the entity on the server.

Properties	Type	Description
Attributes of the entity	mixed	Optional - Values to modify
<code>__DATACLASS</code>	String	Mandatory - Indicates the Dataclass of the entity
<code>__ENTITY</code>	Boolean	Mandatory - True to indicate to the server that the parameter is an entity
<code>__KEY</code>	mixed (same type as the primary key)	Optional - Primary key of the entity

- If `__KEY` is not provided, a new entity is created on the server with the given attributes.
- If **KEY is provided, the entity corresponding to KEY** is loaded on the server with the given attributes

See examples for [creating](#) or [updating](#) entities.

## Related entity parameter

Same properties as for an [entity parameter](#). In addition, the related entity must exist

and is referenced by \_\_KEY containing its primary key.

See examples for [creating](#) or [updating](#) entities with related entities.

## Entity selection parameter

The entity selection must have been defined beforehand using [\\$method=entityset](#).

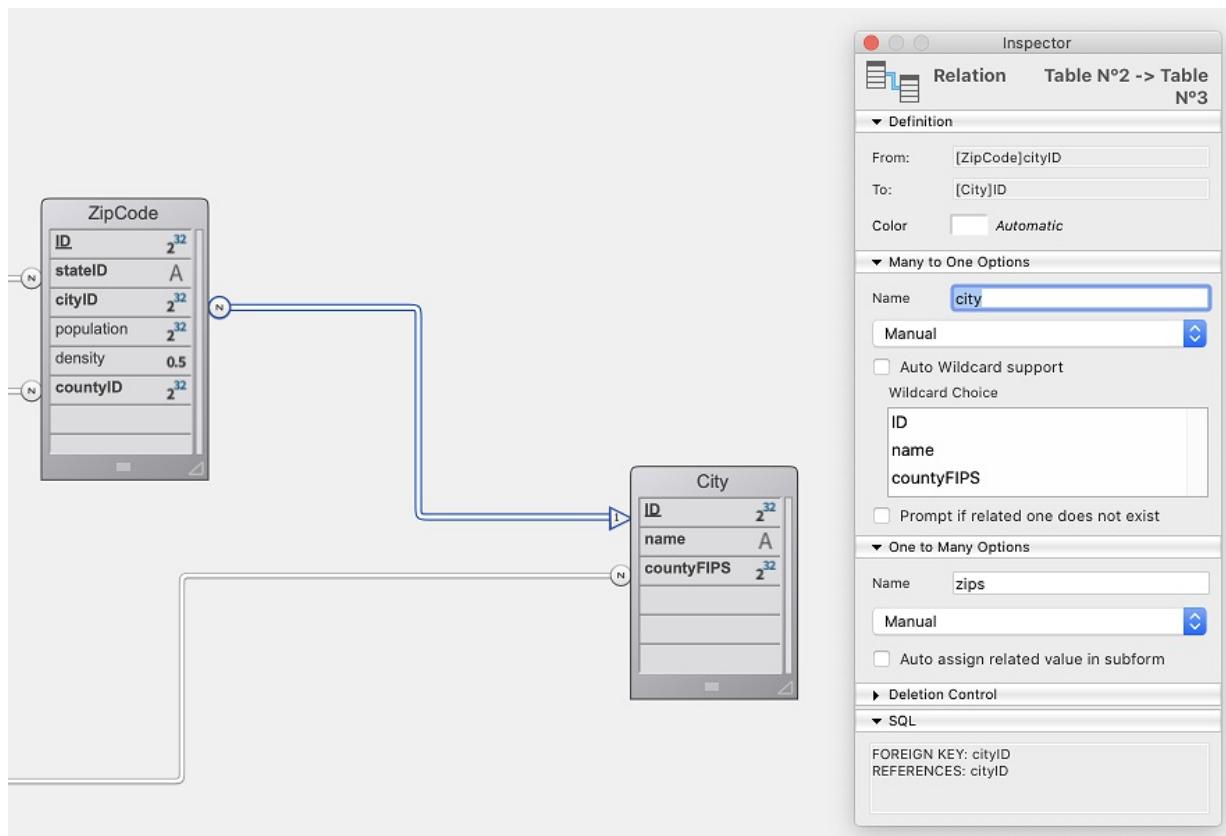
If the request sends a modified entity selection to the server, the called ORDA data model function will be automatically executed on the server with the modified entity selection.

Properties	Type	Description
Attributes of the entity	mixed	Optional - Values to modify
__DATASET	String	Mandatory - entitySetID (UUID) of the entity selection
__ENTITIES	Boolean	Mandatory - True to indicate to the server that the parameter is an entity selection

See example for [receiving an entity selection](#).

## Request examples

This database is exposed as a remote datastore on localhost (port 8111):



## Using a datastore class function

The US\_Cities `DataStore` class provides an API:

```
// DataStore class  
  
Class extends DataStoreImplementation  
  
exposed Function getName()  
    $0:="US cities and zip codes manager"
```

You can then run this request:

**POST** 127.0.0.1:8111/rest/\$catalog/getName

## Result

```
{  
"result": "US cities and zip codes manager"  
}
```

## Using a dataclass class function

The Dataclass class `City` provides an API that returns a city entity from a name passed in parameter:

```
// City class  
  
Class extends DataClass  
  
exposed Function getCity()  
    var $0 : cs.CityEntity  
    var $1,$nameParam : text  
    $nameParam:=$1  
    $0:=This.query("name = :1";$nameParam).first()
```

You can then run this request:

**POST** 127.0.0.1:8111/rest/City/getCity

Body of the request: ["Aguada"]

## Result

The result is an entity:

```
{  
    "__entityModel": "City",  
    "__DATACLASS": "City",  
    "__KEY": "1",  
    "__TIMESTAMP": "2020-03-09T08:03:19.923Z",  
    "__STAMP": 1,  
    "ID": 1,  
    "name": "Aguada",  
    "countyFIPS": 72003,  
    "county": {  
        "__deferred": {  
            "uri": "/rest/County(72003)",  
            "__KEY": "72003"  
        }  
    },  
    "zips": {  
        "__deferred": {  
            "uri": "/rest/City(1)/zips?$expand=zips"  
        }  
    }  
}
```

## Using an entity class function

The Entity class `CityEntity` provides an API:

```
// CityEntity class  
  
Class extends Entity  
  
exposed Function getPopulation()  
    $0:=This.zips.sum("population")
```

You can then run this request:

**POST** `127.0.0.1:8111/rest/City(2)/getPopulation`

## Result

```
{  
    "result": 48814  
}
```

## Using an entitySelection class function

The EntitySelection class `CitySelection` provides an API:

```
// CitySelection class  
  
Class extends EntitySelection  
  
exposed Function getPopulation()  
    $0:=This.zips.sum("population")
```

You can then run this request:

**POST** `127.0.0.1:8111/rest/City/getPopulation/?$filter="ID<3"`

## Result

```
{  
    "result": 87256  
}
```

## Using an entitySelection class function and an entitySet

The `StudentsSelection` class has a `getAgeAverage` function:

```
// StudentsSelection Class  
  
Class extends EntitySelection  
  
exposed Function getAgeAverage  
    C_LONGINT($sum;$0)  
    C_OBJECT($s)  
  
    $sum:=0  
    For each ($s;This)  
        $sum:=$sum+$s.age()  
    End for each  
    $0:=$sum/This.length
```

Once you have created an entityset, you can run this request:

**POST**

`127.0.0.1:8044/rest/Students/getAgeAverage/$entityset/17E83633FFB54ECD  
BF947E5C620BB532`

## Result

```
{  
    "result": 34  
}
```

## Using an entitySelection class function and an orderBy

The `StudentsSelection` class has a `getLastSummary` function:

```
// StudentsSelection Class
```

```
Class extends EntitySelection  
  
exposed Function getLastSummary  
    C_TEXT($0)  
    C_OBJECT($last)  
  
    $last:=This.last()  
    $0:=$last.firstname+" - "+$last.lastname+" is ...  
"+String($last.age())
```

You can then run this request:

```
POST 127.0.0.1:8044/rest/Students/getLastSummary/$entityset/?  
$filter="lastname=b@"&$orderby="lastname"
```

## Result

```
{  
    "result": "Wilbert - Bull is ... 21"  
}
```

## Using an entity to be created on the server

The Dataclass class `Students` has the function `pushData()` receiving an entity containing data from the client. The `checkData()` method runs some controls. If they are OK, the entity is saved and returned.

```
// Students Class  
  
Class extends DataClass  
  
exposed Function pushData  
    var $1, $entity, $status, $0 : Object  
  
    $entity:=$1  
  
    $status:=checkData($entity) // $status is an object with a success  
boolean property  
  
    $0:=$status  
  
    If ($status.success)  
        $status:=$entity.save()  
        If ($status.success)  
            $0:=$entity  
        End if  
    End if
```

You run this request:

**POST** `http://127.0.0.1:8044/rest/Students/pushData`

Body of the request:

```
[ {  
    "__DATACLASS": "Students",  
    "__ENTITY": true,  
    "firstname": "Ann",  
    "lastname": "Brown"  
}]
```

Since no `__KEY` is given, a new Students entity is loaded on the server **with the attributes received from the client**. Because the `pushData()` function runs a `save()` action, the new entity is created.

## Result

```
{  
    "__entityModel": "Students",  
    "__DATACLASS": "Students",  
    "__KEY": "55",  
    "__TIMESTAMP": "2020-06-16T10:54:41.805Z",  
    "__STAMP": 1,  
    "ID": 55,  
    "firstname": "Ann",  
    "lastname": "BROWN",  
    "schoolID": null,  
    "school": null  
}
```

## Using an entity to be updated on the server

Same as above but with a `__KEY` attribute

You run this request:

**POST:** `http://127.0.0.1:8044/rest/Students/pushData`

Body of the request:

```
[ {  
    "__DATACLASS": "Students",  
    "__ENTITY": true,  
    "lastname": "Brownie",  
    "__KEY": 55  
}]
```

Since `__KEY` is given, the Students entity with primary key 55 is loaded **with the lastname value received from the client**. Because the function runs a `save()` action, the entity is updated.

## Result

```
{
    "__entityModel": "Students",
    "__DATACLASS": "Students",
    "__KEY": "55",
    "__TIMESTAMP": "2020-06-16T11:10:21.679Z",
    "__STAMP": 3,
    "ID": 55,
    "firstname": "Ann",
    "lastname": "BROWNIE",
    "schoolID": null,
    "school": null
}
```

## Creating an entity with a related entity

In this example, we create a new Students entity with the Schools entity having primary key 2.

You run this request:

**POST:** `http://127.0.0.1:8044/rest/Students/pushData`

Body of the request:

```
[ {
    "__DATACLASS": "Students",
    "__ENTITY": true,
    "firstname": "John",
    "lastname": "Smith",
    "school": { "__KEY": 2 }
}]
```

## Result

```
{
    "__entityModel": "Students",
    "__DATACLASS": "Students",
    "__KEY": "56",
    "__TIMESTAMP": "2020-06-16T11:16:47.601Z",
    "__STAMP": 1,
    "ID": 56,
    "firstname": "John",
    "lastname": "SMITH",
    "schoolID": 2,
    "school": {
        "__deferred": {
            "uri": "/rest/Schools(2)",
            "__KEY": "2"
        }
    }
}
```

## Updating an entity with a related entity

In this example, we associate an existing school to a Students entity. The `StudentsEntity` class has an API:

```

// StudentsEntity class

Class extends Entity

exposed Function putToSchool()
    var $1, $school , $0, $status : Object

        // $1 is a Schools entity
        $school:=$1
        // Associate the related entity school to the current Students
entity
        This.school:=$school

        $status:=This.save()

        $0:=$status

```

You run this request, called on a Students entity : **POST**

`http://127.0.0.1:8044/rest/Students(1)/putToSchool` Body of the request:

```
[ {
  "DATACLASS": "Schools",
  "ENTITY": true,
  "KEY": 2
}]
```

## Result

```
{
  "result": {
    "success": true
  }
}
```

## Receiving an entity selection as parameter

In the `Students` Dataclass class, the `setFinalExam()` function updates a received entity selection (\$1). It actually updates the `finalExam` attribute with the received value (\$2). It returns the primary keys of the updated entities.

```

// Students class

Class extends DataClass

exposed Function setFinalExam()

    var $1, $es, $student, $status : Object
    var $2, $examResult : Text

    var $keys, $0 : Collection

        //Entity selection
    $es:=$1

    $examResult:=$2

    $keys:=New collection()

        //Loop on the entity selection
    For each ($student;$es)
        $student.finalExam:=$examResult
        $status:=$student.save()
        If ($status.success)
            $keys.push($student.ID)
        End if
    End for each

    $0:=$keys

```

An entity set is first created with this request:

```
http://127.0.0.1:8044/rest/Students/?$filter="ID<3"&$method=entityset
```

Then you can run this request:

```
POST http://127.0.0.1:8044/rest/Students/setFinalExam
```

Body of the request:

```
[
{
    "ENTITIES":true,
    "DATASET":"9B9C053A111E4A288E9C1E48965FE671"
},
"Passed"
]
```

## Result

The entities with primary keys 1 and 2 have been updated.

```
{
    "result": [
        1,
        2
    ]
}
```

## Using an entity selection updated on the client

Using the `getAgeAverage()` function [defined above](#).

```
var $remoteDS, $newStudent, $students : Object
var $ageAverage : Integer

$remoteDS:=Open datastore (New
object ("hostname";"127.0.0.1:8044");"students")

// $newStudent is a student entity to procees
$newStudent:=...
$students:=$remoteDS.Students.query("school.name = :1";"Math school")
// We add an entity to the $students entity selection on the client
$students.add($newStudent)

// We call a function on the StudentsSelection class returning the age
average of the students in the entity selection
// The function is executed on the server on the updated $students
entity selection which included the student added from the client
$ageAverage:=$students.getAgeAverage()
```

## About REST Requests

The following structures are supported for REST requests:

URI	Resource (Input)	/? or &{filter} (Output)
http://{servername}: {port}/rest/	{dataClass}	\$filter, \$attributes, \$skip, \$method=....
	{dataClass}/\$entityset/{entitySetID}	\$method=...
	{dataClass}({key})	\$attributes
	{dataClass}:{attribute}(value)	

While all REST requests must contain the URI and Resource parameters, the Output (which filters the data returned) is optional.

As with all URIs, the first parameter is delimited by a "?" and all subsequent parameters by a "&". For example:

```
GET /rest/Person/?  
$filter="lastName!=Jones"&$method=entityset&$timeout=600
```

You can place all values in quotes in case of ambiguity. For example, in our above example, we could have put the value for the last name in single quotes: "lastName!="Jones"".

The parameters allow you to manipulate data in dataclasses in your 4D project. Besides retrieving data using `GET` HTTP methods, you can also add, update, and delete entities in a dataclass using `POST` HTTP methods.

If you want the data to be returned in an array instead of JSON, use the `$asArray` parameter.

## REST Status and Response

With each REST request, the server returns the status and a response (with or without an error).

### Request Status

With each REST request, you get the status along with the response. Below are a few of the statuses that can arise:

Status	Description
0	Request not processed (server might not be started).
200 OK	Request processed without error.
401 Unauthorized	Permissions error (check user's permissions).
402 No session	Maximum number of sessions has been reached.
404 Not Found	The data class is not accessible via REST or the entity set doesn't exist.
500 Internal Server Error	Error processing the REST request.

### Response

The response (in JSON format) varies depending on the request.

If an error arises, it will be sent along with the response from the server or it will be the response from the server.

# API (general)

REST API for global information

## \$catalog

The catalog describes all the dataclasses and attributes available in the datastore.

## \$directory

The directory handles user access through REST requests.

## \$info

Returns information about the entity sets currently stored in 4D Server's cache as we...

## \$upload

Returns an ID of the file uploaded to the server

## \$catalog

The catalog describes all the dataclasses and attributes available in the datastore.

## Available syntaxes

Syntax	Example	Description
<a href="#"><u>\$catalog</u></a>	/\$catalog	Returns a list of the dataclasses in your project along with two URIs
<a href="#"><u>\$catalog/\$all</u></a>	/\$catalog/\$all	Returns information about all of your project's dataclasses and their attributes
<a href="#"><u>\$catalog/{dataClass}</u></a>	/\$catalog/Employee	Returns information about a dataclass and its attributes

## \$catalog

Returns a list of the dataclasses in your project along with two URIs: one to access the information about its structure and one to retrieve the data in the dataclass

### Description

When you call `$catalog`, a list of the dataclasses is returned along with two URIs for each dataclass in your project's datastore.

Only the exposed dataclasses are shown in this list for your project's datastore. For more information, please refer to [Exposing tables and fields](#) section.

Here is a description of the properties returned for each dataclass in your project's datastore:

Property Type	Description
name	String Name of the dataclass.
uri	String A URI allowing you to obtain information about the dataclass and its attributes.
dataURI	String A URI that allows you to view the data in the dataclass.

### Example

```
GET /rest/$catalog
```

#### Result:

```
{
  dataClasses: [
    {
      name: "Company",
      uri: "http://127.0.0.1:8081/rest/$catalog/Company",
      dataURI: "http://127.0.0.1:8081/rest/Company"
    },
    {
      name: "Employee",
      uri: "http://127.0.0.1:8081/rest/$catalog/Employee",
      dataURI: "http://127.0.0.1:8081/rest/Employee"
    }
  ]
}
```

## \$catalog/\$all

Returns information about all of your project's dataclasses and their attributes

### Description

Calling `$catalog/$all` allows you to receive detailed information about the attributes in each of the dataclasses in your project's active model.

For more information about what is returned for each dataclass and its attributes, use [`\$catalog/{dataClass}`](#).

### Example

```
GET /rest/$catalog/$all
```

#### Result:

```
{  
    "dataClasses": [  
        {  
            "name": "Company",  
            "className": "Company",  
            "collectionName": "CompanySelection",  
            "tableNumber": 2,  
            "scope": "public",  
            "dataURI": "/rest/Company",  
            "attributes": [  
                {  
                    "name": "ID",  
                    "kind": "storage",  
                    "fieldPos": 1,  
                    "scope": "public",  
                    "indexed": true,  
                    "type": "long",  
                    "identifying": true  
                },  
                {  
                    "name": "name",  
                    "kind": "storage",  
                    "fieldPos": 2,  
                    "scope": "public",  
                    "type": "string"  
                },  
                {  
                    "name": "revenues",  
                    "kind": "storage",  
                    "fieldPos": 3,  
                    "scope": "public",  
                    "type": "number"  
                },  
                {  
                    "name": "staff",  
                    "kind": "relatedEntities",  
                    "fieldPos": 4,  
                    "scope": "public",  
                    "type": "EmployeeSelection",  
                    "reversePath": true,  
                    "path": "employer"  
                },  
                {  
                    "name": "url",  
                    "kind": "storage",  
                    "fieldPos": 5,  
                    "scope": "public",  
                    "type": "string"  
                }  
            ]  
        }  
    ]  
}
```

```

        "scope": "public",
        "type": "string"
    }
],
"key": [
{
    "name": "ID"
}
]
},
{
    "name": "Employee",
    "className": "Employee",
    "collectionName": "EmployeeSelection",
    "tableNumber": 1,
    "scope": "public",
    "dataURI": "/rest/Employee",
    "attributes": [
{
    "name": "ID",
    "kind": "storage",
    "scope": "public",
    "indexed": true,
    "type": "long",
    "identifying": true
},
{
    "name": "firstname",
    "kind": "storage",
    "scope": "public",
    "type": "string"
},
{
    "name": "lastname",
    "kind": "storage",
    "scope": "public",
    "type": "string"
},
{
    "name": "employer",
    "kind": "relatedEntity",
    "scope": "public",
    "type": "Company",
    "path": "Company"
}
],
"key": [
{
    "name": "ID"
}
]
}
]
```

## \$catalog/{dataClass}

Returns information about a dataclass and its attributes

### Description

Calling `$catalog/{dataClass}` for a specific dataclass will return the following information about the dataclass and the attributes it contains. If you want to retrieve this information for all the dataclasses in your project's datastore, use

`$catalog/$all`.

The information you retrieve concerns the following:

- Dataclass
- Attribute(s)
- Method(s) if any
- Primary key

## DataClass

The following properties are returned for an exposed dataclass:

Property	Type	Description
name	String	Name of the dataclass
collectionName	String	Name of an entity selection on the dataclass
tableNumber	Number	Table number in the 4D database
scope	String	Scope for the dataclass (note that only dataclasses whose <b>Scope</b> is public are displayed)
dataURI	String	A URI to the data in the dataclass

## Attribute(s)

Here are the properties for each exposed attribute that are returned:

Property	Type	Description
name	String	Attribute name.
kind	String	Attribute type (storage or relatedEntity).
fieldPos	Number	Position of the field in the database table).
scope	String	Scope of the attribute (only those attributes whose scope is Public will appear).
indexed	String	If any <b>Index Kind</b> was selected, this property will return true. Otherwise, this property does not appear.
type	String	Attribute type (bool, blob, byte, date, duration, image, long, long64, number, string, uuid, or word) or the dataclass for a N->1 relation attribute.
identifying	Boolean	This property returns True if the attribute is the primary key. Otherwise, this property does not appear.
path	String	Name of the dataclass for a relatedEntity attribute, or name of the relation for a relatedEntities attribute.
foreignKey	String	For a relatedEntity attribute, name of the related attribute.
inverseName	String	Name of the opposite relation for a relatedEntity or relateEntities attribute.

## Primary Key

The key object returns the **name** of the attribute defined as the **Primary Key** for the dataclass.

## Example

You can retrieve the information regarding a specific dataclass.

`GET /rest/$catalog/Employee`

## Result:

```
{  
    name: "Employee"
```

```
name: "Employee",
className: "Employee",
collectionName: "EmployeeCollection",
scope: "public",
dataURI: "http://127.0.0.1:8081/rest/Employee",
defaultTopSize: 20,
extraProperties: {
    panelColor: "#76923C",
    __CDATA: "\n\n\t\t\n",
    panel: {
        isOpen: "true",
        pathVisible: "true",
        __CDATA: "\n\n\t\t\t\t\n",
        position: {
            X: "394",
            Y: "42"
        }
    }
},
attributes: [
    {
        name: "ID",
        kind: "storage",
        scope: "public",
        indexed: true,
        type: "long",
        identifying: true
    },
    {
        name: "firstName",
        kind: "storage",
        scope: "public",
        type: "string"
    },
    {
        name: "lastName",
        kind: "storage",
        scope: "public",
        type: "string"
    },
    {
        name: "fullName",
        kind: "calculated",
        scope: "public",
        type: "string",
        readOnly: true
    },
    {
        name: "salary",
        kind: "storage",
        scope: "public",
        type: "number",
        defaultFormat: {
            format: "$###,###.00"
        }
    },
    {
        name: "photo",
        kind: "storage",
        scope: "public",
        type: "image"
    },
    {
        name: "employer",
        kind: "relatedEntity",
        scope: "public",
        type: "Employee"
    }
]
```

```
        type: "Company",
        path: "Company"
    },
{
    name: "employerName",
    kind: "alias",
    scope: "public",

    type: "string",
    path: "employer.name",
    readOnly: true
},
{
    name: "description",
    kind: "storage",
    scope: "public",
    type: "string",
    multiLine: true
},
],
key: [
{
    name: "ID"
}
]
}
```

## \$directory

The directory handles user access through REST requests.

### \$directory/login

Opens a REST session on your 4D application and logs in the user.

#### Description

Use `$directory/login` to open a session in your 4D application through REST and login a user. You can also modify the default 4D session timeout.

All parameters must be passed in **headers** of a POST method:

Header key	Header value
username-4D	User - Not mandatory
password-4D	Password - Not mandatory
hashed-password-4D	Hashed password - Not mandatory
session-4D-length	Session inactivity timeout (minutes). Cannot be less than 60 - Not mandatory

#### Example

```
C_TEXT($response;$body_t)
ARRAY TEXT($hKey;3)
ARRAY TEXT($hValues;3)
$hKey{1} := "username-4D"
$hKey{2} := "hashed-password-4D"
$hKey{3} := "session-4D-length"
$hValues{1} := "john"
$hValues{2} := Generate digest("123";4D digest)
$hValues{3} := 120
$httpStatus := HTTP Request (HTTP POST
method;"app.example.com:9000/rest/$directory/login";$body_t;$response;$
```

#### Result:

If the login was successful, the result will be:

```
{
    "result": true
}
```

Otherwise, the response will be:

```
{
    "result": false
}
```

## \$info

Returns information about the entity sets currently stored in 4D Server's cache as well as user sessions

## Description

When you call this request for your project, you retrieve information in the following properties:

Property	Type	Description
cacheSize	Number	4D Server's cache size.
usedCache	Number	How much of 4D Server's cache has been used.
entitySetCount	Number	Number of entity selections.
entitySet	Collection	A collection in which each object contains information about each entity selection.
ProgressInfo	Collection	A collection containing information about progress indicator information.
sessionInfo	Collection	A collection in which each object contains information about each user session.

## entitySet

For each entity selection currently stored in 4D Server's cache, the following information is returned:

Property	Type	Description
id	String	A UUID that references the entity set.
dataClass	String	Name of the dataclass.
selectionSize	Number	Number of entities in the entity selection.
sorted	Boolean	Returns true if the set was sorted (using <code>\$orderby</code> ) or false if it's not sorted.
refreshed	Date	When the entity set was created or the last time it was used. When the entity set will expire (this date/time changes each time when the entity set is refreshed). The difference between
expires	Date	refreshed and expires is the timeout for an entity set. This value is either two hours by default or what you defined using <code>\$timeout</code> .

For information about how to create an entity selection, refer to `$method=entityset`. If you want to remove the entity selection from 4D Server's cache, use `$method=release`.

4D also creates its own entity selections for optimization purposes, so the ones you create with `$method=entityset` are not the only ones returned.

**IMPORTANT** If your project is in **Controlled Admin Access Mode**, you must first log into the project as a user in the Admin group.

## sessionInfo

For each user session, the following information is returned in the `sessionInfo` collection:

<b>Property</b>	<b>Type</b>	<b>Description</b>
sessionID	String	A UUID that references the session.
userNameString		The name of the user who runs the session.
lifeTime	Number	The lifetime of a user session in seconds (3600 by default).
expiration	Date	The current expiration date and time of the user session.

## Example

Retrieve information about the entity sets currently stored in 4D Server's cache as well as user sessions:

```
GET /rest/$info
```

**Result:**

The progress indicator information listed after the entity selections is used internally by 4D.



## \$upload

Returns an ID of the file uploaded to the server

### Description

Post this request when you have a file that you want to upload to the Server. If you have an image, you pass `$rawPict=true`. For all other files, you pass `$binary=true`.

You can modify the timeout, which by default is 120 seconds, by passing a value to the `$timeout` parameter.

### Uploading scenario

Imagine you want to upload an image to update the picture attribute of an entity.

To upload an image (or any binary file), you must first select the file from the client application. The file itself must be passed in the **body** of the request.

Then, you upload the selected image to 4D Server using a request such as:

```
POST /rest/$upload?${$rawPict=true}
```

As a result, the server returns an ID that identifies the file:

#### Response:

```
{ "ID": "D507BC03E613487E9B4C2F6A0512FE50" }
```

Afterwards, you use this ID to add it to an attribute using `[$method=update]` to add the image to an entity. The request looks like:

```
POST /rest/Employee/?${$method=update}
```

#### POST data:

```
{
  KEY: "12",
  STAMP: 4,
  photo: { "ID": "D507BC03E613487E9B4C2F6A0512FE50" }
}
```

#### Response:

The modified entity is returned:

```

{
    "__KEY": "12",
    "__STAMP": 5,
    "uri": "http://127.0.0.1:8081/rest/Employee(12)",
    "ID": 12,
    "firstName": "John",
    "firstName": "Smith",
    "photo":
    {
        "__deferred":
        {
            "uri": "/rest/Employee(12)/photo?
$imageformat=best&$version=1&$expand=photo",
            "image": true
        }
    }
}

```

## Example with a 4D HTTP client

The following example shows how to upload a *.pdf* file to the server using the 4D HTTP client.

```

var $params : Text
var $response : Object
var $result : Integer
var $blob : Blob

ARRAY TEXT($headerNames; 1)
ARRAY TEXT($headerValues; 1)

$url:="localhost:80/rest/$upload?$binary=true" //prepare the REST
request

$headerNames{1}:="Content-Type"
$headerValues{1}:="application/octet-stream"

DOCUMENT TO BLOB("c:\\invoices\\inv003.pdf"; $blob) //Load the binary

//Execute the first POST request to upload the file
$result:=HTTP Request(HTTP POST method; $url; $blob; $response;
$headerNames; $headerValues)

If ($result=200)
    var $data : Object
    $data:=New object
    $data.__KEY:="3"
    $data.__STAMP:="3"
    $data.pdf:=New object("ID"; String($response.ID))

    $url:="localhost:80/rest/Invoices?$method=update" //second request
    to update the entity

    $headerNames{1}:="Content-Type"
    $headerValues{1}:="application/json"

    $result:=HTTP Request(HTTP POST method; $url; $data; $response;
$headerNames; $headerValues)
Else
    ALERT(String($result)+" Error")
End if

```



# API (dataClass)

REST API for dataClass.

## **dataClass**

Dataclass names can be used directly in the REST requests to work with entities and ...

## **\$asArray**

Returns the result of a query in an array (i.e. a collection) instead of a JSON object.

## **\$atomic/\$atOnce**

Allows the actions in the REST request to be in a transaction. If there are no errors, t...

## **\$attributes**

Allows selecting the related attribute(s) to get from the dataclass (e.g., Company(1)?...

## **\$binary**

Pass "true" to save the BLOB as a document (must also pass \$expand=)

## **\$compute**

Calculate on specific attributes (e.g., Employee/salary/?\$compute=sum) or in the ca...

## **\$distinct**

Returns the distinct values for a specific attribute in a collection (e.g., Company/nam...

## **\$entityset**

After creating an entity set by using \$method=entityset, you can then use it subsequ…

## **\$expand**

Expands an image stored in an Image attribute (e.g., Employee(1)/photo?\$imagefor…

## **\$filter**

Allows to query the data in a dataclass or method (e.g., \$filter="firstName!=" AND s…

## **\$imageformat**

Defines which image format to use for retrieving images (e.g., \$imageformat=png)

## **\$lock**

Locks and unlocks an entity using the pessimistic mechanism.

## **\$method**

This parameter allows you to define the operation to execute with the returned entity…

## **\$orderby**

Sorts the data returned by the attribute and sorting order defined (e.g., \$orderby="l…

## **\$querypath**

Returns the query as it was executed by 4D Server (e.g., \$querypath=true)

## **\$queryplan**

Returns the query as it was passed to 4D Server (e.g., \$queryplan=true)

## **\$savedfilter**

Saves the filter defined by \$filter when creating an entity set (e.g., \$savedfilter="")

## **\$savedorderby**

Saves the order by defined by \$orderby when creating an entity set (e.g., \$savedord...

## **\$skip**

Starts the entity defined by this number in the collection (e.g., \$skip=10)

## **\$timeout**

Defines the number of seconds to save an entity set in 4D Server's cache (e.g., \$time...

## **\$top/\$limit**

Limits the number of entities to return (e.g., \$top=50)

## **\$version**

Image version number

## dataClass

Dataclass names can be used directly in the REST requests to work with entities and entity selections, or class functions of the dataclass.

### Available syntaxes

Syntax	Example	Description
<a href="#"><u>{dataClass}</u></a>	/Employee	Returns the dataclass name by default first 100 entities of the dataclass
<a href="#"><u>{dataClass}[{key}]</u></a>	/Employee[22]	Returns data for specific entity defined by the key in the dataclass primary key
<a href="#"><u>{dataClass}:{attribute}(value)</u></a>	/Employee:firstName(John)	Returns data for entity if attribute value is defined
<a href="#"><u>{dataClass}/{DataClassClassFunction}</u></a>	/City/getCity	Execute dataclass class function
<a href="#"><u>{dataClass}/{EntitySelectionClassFunction}</u></a>	/City/getPopulation/?\$filter="ID<3"	Execute entity selection class function
<a href="#"><u>{dataClass}[{key}]/{EntityClassFunction}</u></a>	City[2]/getPopulation	Execute entity class function

Function calls are detailed in the [Calling ORDA class functions](#) section.

### [{dataClass}](#)

Returns all the data (by default the first 100 entities) for a specific dataclass (e.g., Company)

#### Description

When you call this parameter in your REST request, the first 100 entities are returned

unless you have specified a value using `$stop/$limit`.

Here is a description of the data returned:

Property	Type	Description
<code>__entityModel</code>	String	Name of the dataclass.
<code>__COUNT</code>	Number	Number of entities in the dataclass.
<code>__SENT</code>	Number	Number of entities sent by the REST request. This number can be the total number of entities if it is less than the value defined by <code>\$stop/\$limit</code> .
<code>__FIRST</code>	Number	Entity number that the selection starts at. Either 0 by default or the value defined by <code>\$skip</code> .
<code>__ENTITIES</code>	Collection	This collection of objects contains an object for each entity with all its attributes. All relational attributes are returned as objects with a URI to obtain information regarding the parent.

Each entity contains the following properties:

Property	Type	Description
<code>__KEY</code>	String	Value of the primary key defined for the dataclass.
<code>__TIMESTAMP</code>	Date	Timestamp of the last modification of the entity
<code>__STAMP</code>	Number	Internal stamp that is needed when you modify any of the values in the entity when using <code>\$method=update</code> .

If you want to specify which attributes you want to return, define them using the following syntax `{attribute1, attribute2, ...}`. For example:

```
GET /rest/Company/name,address
```

## Example

Return all the data for a specific dataclass.

```
GET /rest/Company
```

### Result:

```
{
    "__entityModel": "Company",
    "__GlobalStamp": 51,
    "__COUNT": 250,
    "__SENT": 100,
    "__FIRST": 0,
    "__ENTITIES": [
        {
            "__KEY": "1",
            "__TIMESTAMP": "2020-04-10T10:44:49.927Z",
            "__STAMP": 1,
            "ID": 1,
            "name": "Adobe",
            "address": null,
            "city": "San Jose",
            "country": "USA",
            "revenues": 500000,
            "staff": {
                "__deferred": {
                    "uri": "http://127.0.0.1:8081/rest/Company(1)/staff?$expand=staff"
                }
            }
        },
    ],
}
```

```

{
    "__KEY": "2",
    "__TIMESTAMP": "2018-04-25T14:42:18.351Z",
    "__STAMP": 1,
    "ID": 2,
    "name": "Apple",
    "address": null,
    "city": "Cupertino",
    "country": "USA",
    "revenues": 890000,
    "staff": {
        "__deferred": {
            "uri": "http://127.0.0.1:8081/rest/Company(2)/staff?$expand=staff"
        }
    }
},
{
    "__KEY": "3",
    "__TIMESTAMP": "2018-04-23T09:03:49.021Z",
    "__STAMP": 2,
    "ID": 3,
    "name": "4D",
    "address": null,
    "city": "Clichy",
    "country": "France",
    "revenues": 700000,
    "staff": {
        "__deferred": {
            "uri": "http://127.0.0.1:8081/rest/Company(3)/staff?$expand=staff"
        }
    }
},
{
    "__KEY": "4",
    "__TIMESTAMP": "2018-03-28T14:38:07.430Z",
    "__STAMP": 1,
    "ID": 4,
    "name": "Microsoft",
    "address": null,
    "city": "Seattle",
    "country": "USA",
    "revenues": 650000,
    "staff": {
        "__deferred": {
            "uri": "http://127.0.0.1:8081/rest/Company(4)/staff?$expand=staff"
        }
    }
}
....//more entities here
]
}

```

## {dataClass}[{key}]

Returns the data for the specific entity defined by the dataclass's primary key, e.g., `Company[22]` or `Company[IT0911AB2200]`

### Description

By passing the dataclass and a key, you can retrieve all the public information for that entity. The key is the value in the attribute defined as the Primary Key for your

dataclass. For more information about defining a primary key, refer to the **Modifying the Primary Key** section in the **Data Model Editor**.

For more information about the data returned, refer to [{DataClass}](#).

If you want to specify which attributes you want to return, define them using the following syntax [{attribute1, attribute2, ...}](#). For example:

```
GET /rest/Company[1]/name,address
```

If you want to expand a relation attribute using `$expand`, you do so by specifying it as shown below:

```
GET /rest/Company[1]/name,address,staff?$expand=staff
```

## Example

The following request returns all the public data in the Company dataclass whose key is 1.

```
GET /rest/Company[1]
```

### Result:

```
{
  "__entityModel": "Company",
  "__KEY": "1",
  "__TIMESTAMP": "2020-04-10T10:44:49.927Z",
  "__STAMP": 2,
  "ID": 1,
  "name": "Apple",
  "address": Infinite Loop,
  "city": "Cupertino",
  "country": "USA",
  "url": http://www.apple.com,
  "revenues": 500000,
  "staff": {
    "__deferred": {
      "uri": "http://127.0.0.1:8081/rest/Company(1)/staff?
$expand=staff"
    }
  }
}
```

## {dataClass}:{attribute}(value)

Returns the data for one entity in which the attribute's value is defined

### Description

By passing the *dataClass* and an *attribute* along with a value, you can retrieve all the public information for that entity. The value is a unique value for attribute, but is not the primary key.

```
GET /rest/Company:companyCode(Acme001)
```

If you want to specify which attributes you want to return, define them using the following syntax [{attribute1, attribute2, ...}](#). For example:

```
GET /rest/Company:companyCode(Acme001)/name,address
```

If you want to use a relation attribute using `$attributes`, you do so by specifying it as

shown below:

```
GET /rest/Company:companyCode(Acme001)?  
$attributes=name,address,staff.name
```

## Example

The following request returns all the public data of the employee named "Jones".

```
GET /rest/Employee:lastname(Jones)
```

## \$asArray

Returns the result of a query in an array (i.e. a collection) instead of a JSON object.

### Description

If you want to receive the response in an array, you just have to add `$asArray` to your REST request (e.g., `$asArray=true`).

### Example

Here is an example of how to receive the response in an array.

```
GET /rest/Company/?$filter="name begin a"&$top=3&$asArray=true
```

#### Response:

```
[  
  {  
    "__KEY": 15,  
    "__STAMP": 0,  
    "ID": 15,  
    "name": "Alpha North Yellow",  
    "creationDate": "!!0000-00-00!!",  
    "revenues": 82000000,  
    "extra": null,  
    "comments": "",  
    "__GlobalStamp": 0  
  },  
  {  
    "__KEY": 34,  
    "__STAMP": 0,  
    "ID": 34,  
    "name": "Astral Partner November",  
    "creationDate": "!!0000-00-00!!",  
    "revenues": 90000000,  
    "extra": null,  
    "comments": "",  
    "__GlobalStamp": 0  
  },  
  {  
    "__KEY": 47,  
    "__STAMP": 0,  
    "ID": 47,  
    "name": "Audio Production Uniform",  
    "creationDate": "!!0000-00-00!!",  
    "revenues": 28000000,  
    "extra": null,  
    "comments": "",  
    "__GlobalStamp": 0  
  }  
]
```

The same data in its default JSON format:

```

{
    "__entityModel": "Company",
    "__GlobalStamp": 50,
    "__COUNT": 52,
    "__FIRST": 0,
    "__ENTITIES": [
        {
            "__KEY": "15",
            "__TIMESTAMP": "2018-03-28T14:38:07.434Z",
            "__STAMP": 0,
            "ID": 15,
            "name": "Alpha North Yellow",
            "creationDate": "0!0!0",
            "revenues": 82000000,
            "extra": null,
            "comments": "",
            "__GlobalStamp": 0,
            "employees": {
                "__deferred": {
                    "uri": "/rest/Company(15)/employees?
$expand=employees"
                }
            }
        },
        {
            "__KEY": "34",
            "__TIMESTAMP": "2018-03-28T14:38:07.439Z",
            "__STAMP": 0,
            "ID": 34,
            "name": "Astral Partner November",
            "creationDate": "0!0!0",
            "revenues": 90000000,
            "extra": null,
            "comments": "",
            "__GlobalStamp": 0,
            "employees": {
                "__deferred": {
                    "uri": "/rest/Company(34)/employees?
$expand=employees"
                }
            }
        },
        {
            "__KEY": "47",
            "__TIMESTAMP": "2018-03-28T14:38:07.443Z",
            "__STAMP": 0,
            "ID": 47,
            "name": "Audio Production Uniform",
            "creationDate": "0!0!0",
            "revenues": 28000000,
            "extra": null,
            "comments": "",
            "__GlobalStamp": 0,
            "employees": {
                "__deferred": {
                    "uri": "/rest/Company(47)/employees?
$expand=employees"
                }
            }
        }
    ],
    "__SENT": 3
}

```



# Page Not Found

We could not find what you were looking for.

Please contact the owner of the site that linked you to the original URL and let them know their link is broken.

## \$attributes

Allows selecting the related attribute(s) to get from the dataclass (e.g., `Company(1)?$attributes=employees.lastname` or `Employee?$attributes=employer.name`).

## Description

When you have relation attributes in a dataclass, use `$attributes` to define the path of attributes whose values you want to get for the related entity or entities.

You can apply `$attributes` to an entity (e.g., `People(1)`) or an entity selection (e.g., `People/$entityset/0AF4679A5C394746BFEB68D2162A19FF`).

- If `$attributes` is not specified in a query, or if the "\*" value is passed, all available attributes are extracted. **Related entity** attributes are extracted with the simple form: an object with property `__KEY` (primary key) and `URI`. **Related entities** attributes are not extracted.
- If `$attributes` is specified for **related entity** attributes:
  - `$attributes=relatedEntity`: the related entity is returned with simple form (deferred `__KEY` property (primary key)) and `URI`.
  - `$attributes=relatedEntity.*`: all the attributes of the related entity are returned
  - `$attributes=relatedEntity.attributePath1, relatedEntity.attributePath2, ...`: only those attributes of the related entity are returned.
- If `$attributes` is specified for **related entities** attributes:
  - `$attributes=relatedEntities.*`: all the properties of all the related entities are returned
  - `$attributes=relatedEntities.attributePath1, relatedEntities.attributePath2, ...`: only those attributes of the related entities are returned.

## Example with related entities

If we pass the following REST request for our Company dataclass (which has a relation attribute "employees"):

```
GET /rest/Company(1)/?$attributes=employees.lastname
```

**Response:**

```
{
    "__entityModel": "Company",
    "__KEY": "1",
    "__TIMESTAMP": "2018-04-25T14:41:16.237Z",
    "__STAMP": 2,
    "employees": {
        "__ENTITYSET": "/rest/Company(1)/employees?$expand=employees",
        "__GlobalStamp": 50,
        "__COUNT": 135,
        "__FIRST": 0,
        "__ENTITIES": [
            {
                "__KEY": "1",
                "__TIMESTAMP": "2019-12-01T20:18:26.046Z",
                "__STAMP": 5,
                "lastname": "ESSEAL"
            },
            {
                "__KEY": "2",
                "__TIMESTAMP": "2019-12-04T10:58:42.542Z",
                "__STAMP": 6,
                "lastname": "JONES"
            },
            ...
        ]
    }
}
```

If you want to get all attributes from employees:

```
GET /rest/Company(1) /?$attributes=employees.*
```

If you want to get last name and job name attributes from employees:

```
GET /rest/Company(1) /?$attributes=employees.lastname,employees.jobname
```

## Example with related entity

If we pass the following REST request for our Employee dataclass (which has several relation attributes, including "employer"):

```
GET /rest/Employee(1) ?$attributes=employer.name
```

**Response:**

```
{
    "__entityModel": "Employee",
    "__KEY": "1",
    "__TIMESTAMP": "2019-12-01T20:18:26.046Z",
    "__STAMP": 5,
    "employer": {
        "__KEY": "1",
        "__TIMESTAMP": "2018-04-25T14:41:16.237Z",
        "__STAMP": 0,
        "name": "Adobe"
    }
}
```

If you want to get all attributes of the employer:

```
GET /rest/Employee(1) ?$attributes=employer.*
```

If you want to get the last names of all employees of the employer:

```
GET /rest/Employee(1)?$attributes=employer.employees.lastname
```

## \$binary

Pass "true" to save the BLOB as a document (must also pass `$expand={blobAttributeName}`)

### Description

`$binary` allows you to save the BLOB as a document. You must also use the [`\$expand`](#) command in conjunction with it.

When you make the following request:

```
GET /rest/Company(11)/blobAtt?$binary=true&$expand=blobAtt
```

You will be asked where to save the BLOB to disk:

## \$compute

Calculate on specific attributes (e.g., Employee/salary/?\$compute=sum) or in the case of an Object attribute (e.g., Employee/objectAtt.property1/?\$compute=sum)

## Description

This parameter allows you to do calculations on your data.

If you want to perform a calculation on an attribute, you write the following:

```
GET /rest/Employee/salary/?$compute=$all
```

If you want to pass an Object attribute, you must pass one of its property. For example:

```
GET /rest/Employee/objectAtt.property1/?$compute=$all
```

You can use any of the following keywords:

Keyword	Description
\$all	A JSON object that defines all the functions for the attribute (average, count, min, max, and sum for attributes of type Number and count, min, and max for attributes of type String)
average	Get the average on a numerical attribute
count	Get the total number in the collection or dataclass (in both cases you must specify an attribute)
min	Get the minimum value on a numerical attribute or the lowest value in an attribute of type String
max	Get the maximum value on a numerical attribute or the highest value in an attribute of type String
sum	Get the sum on a numerical attribute

## Example

If you want to get all the computations for an attribute of type Number, you can write:

```
GET /rest/Employee/salary/?$compute=$all
```

### Response:

```
{
  "salary": {
    "count": 4,
    "sum": 335000,
    "average": 83750,
    "min": 70000,
    "max": 99000
  }
}
```

If you want to get all the computations for an attribute of type String, you can write:

```
GET /rest/Employee/firstName/?$compute=$all
```

### Response:

```
{  
    "salary": {  
        "count": 4,  
        "min": Anne,  
        "max": Victor  
    }  
}
```

If you want to just get one calculation on an attribute, you can write the following:

```
GET /rest/Employee/salary/?$compute=sum
```

**Response:**

```
235000
```

If you want to perform a calculation on an Object attribute, you can write the following:

```
GET /rest/Employee/objectAttribute.property1/?$compute=sum
```

**Response:**

```
45
```

## \$distinct

Returns the distinct values for a specific attribute in a collection (e.g., `Company/name? $filter="name=a*" &$distinct=true`)

### Description

`$distinct` allows you to return a collection containing the distinct values for a query on a specific attribute. Only one attribute in the dataclass can be specified. Generally, the String type is best; however, you can also use it on any attribute type that could contain multiple values.

You can also use `$skip` and `$top/$limit` as well, if you'd like to navigate the selection before it's placed in an array.

### Example

In our example below, we want to retrieve the distinct values for a company name starting with the letter "a":

```
GET /rest/Company/name?$filter="name=a*" &$distinct=true
```

#### Response:

```
[  
    "Adobe",  
    "Apple"  
]
```

## \$entityset

After [creating an entity set](#) by using `$method=entityset`, you can then use it subsequently.

### Available syntaxes

Syntax	Example	Description
<a href="#"><b>\$entityset/{entitySetID}</b></a>	/People/\$entityset/0ANUMBER	Retrieves an existing entity set
<a href="#"><b>\$entityset/{entitySetID}?</b></a> <a href="#"><b>\$operator...&amp;\$otherCollection</b></a>	/Employee/\$entityset/0ANUMBER? \$logicOperator=AND &\$otherCollection=C0ANUMBER	Creates a new entity set from comparing existing entity sets

## \$entityset/{entitySetID}

Retrieves an existing entity set (e.g.,

```
People/$entityset/0AF4679A5C394746BFEB68D2162A19FF)
```

### Description

This syntax allows you to execute any operation on a defined entity set.

Because entity sets have a time limit on them (either by default or after calling `$timeout` with your own limit), you can call `$savedfilter` and `$savedorderby` to save the filter and order by statements when you create an entity set.

When you retrieve an existing entity set stored in 4D Server's cache, you can also apply any of the following to the entity set: [`\$expand`](#), [`\$filter`](#), [`\$orderby`](#), [`\$skip`](#), and [`\$stop/\$limit`](#).

### Example

After you create an entity set, the entity set ID is returned along with the data. You call this ID in the following manner:

```
GET /rest/Employee/$entityset/9718A30BF61343C796345F3BE5B01CE7
```

## \$entityset/{entitySetID}?\$operator...&\$otherCollection

Create another entity set based on previously created entity sets

Parameter	Type	Description
\$operator	String	One of the logical operators to test with the other entity set
\$otherCollection	String	Entity set ID

### Description

After creating an entity set (entity set #1) by using `$method=entityset`, you can then create another entity set by using the `$entityset/{entitySetID}?` `$operator... &$otherCollection` syntax, the `$operator` property (whose values

are shown below), and another entity set (entity set #2) defined by the `$otherCollection` property. The two entity sets must be in the same dataclass.

You can then create another entity set containing the results from this call by using the `$method=entityset` at the end of the REST request.

Here are the logical operators:

Operator	Description
AND	Returns the entities in common to both entity sets
OR	Returns the entities in both entity sets
EXCEPT	Returns the entities in entity set #1 minus those in entity set #2
INTERSECT	Returns either true or false if there is an intersection of the entities in both entity sets (meaning that least one entity is common in both entity sets)

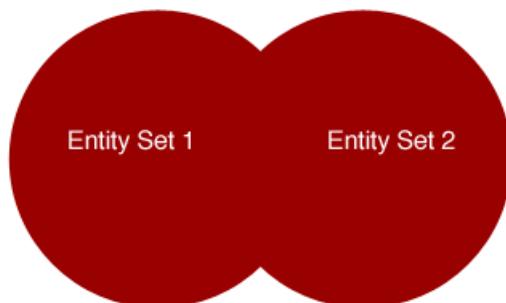
The logical operators are not case-sensitive, so you can write "AND" or "and".

Below is a representation of the logical operators based on two entity sets. The red section is what is returned.

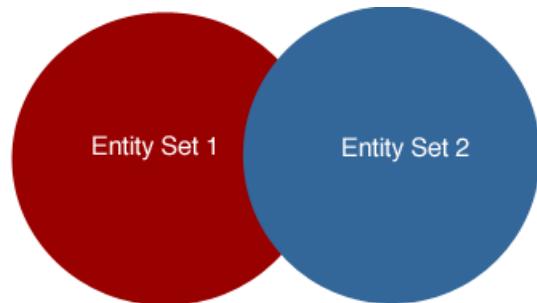
### AND



### OR



### EXCEPT



The syntax is as follows:

```
GET /rest/dataClass/$entityset/entitySetID?  
$logicOperator=AND&$otherCollection=entitySetID
```

## Example

In the example below, we return the entities that are in both entity sets since we are using the AND logical operator:

```
GET /rest/Employee/$entityset/9718A30BF61343C796345F3BE5B01CE7?  
$logicOperator=AND&$otherCollection=C05A0D887C664D4DA1B38366DD21629B
```

If we want to know if the two entity sets intersect, we can write the following:

```
GET /rest/Employee/$entityset/9718A30BF61343C796345F3BE5B01CE7?  
$logicOperator=intersect&$otherCollection=C05A0D887C664D4DA1B38366DD21  
629B
```

If there is an intersection, this query returns true. Otherwise, it returns false.

In the following example we create a new entity set that combines all the entities in both entity sets:

```
GET /rest/Employee/$entityset/9718A30BF61343C796345F3BE5B01CE7?  
$logicOperator=OR&$otherCollection=C05A0D887C664D4DA1B38366DD21629B&$m  
ethod=entityset
```

## \$expand

Expands an image stored in an Image attribute (e.g., `Employee(1)/photo? $imageformat=best&$expand=photo`)

or

Expands an BLOB attribute to save it.

**Compatibility:** For compatibility reasons, \$expand can be used to expand a relational attribute (e.g., `Company(1)?$expand=staff` or `Employee/? $filter="firstName BEGIN a"&$expand=employer`). It is however recommended to use [\\$attributes](#) for this feature.

## Viewing an image attribute

If you want to view an image attribute in its entirety, write the following:

```
GET /rest/Employee(1)/photo? $imageformat=best&$version=1&$expand=photo
```

For more information about the image formats, refer to [\\$imageformat](#). For more information about the version parameter, refer to [\\$version](#).

## Saving a BLOB attribute to disk

If you want to save a BLOB stored in your dataclass, you can write the following by also passing "true" to \$binary:

```
GET /rest/Company(11)/blobAtt? $binary=true&$expand=blobAtt
```

## \$filter

Allows to query the data in a dataclass or method (e.g., `$filter="firstName != '' AND salary>30000"`)

## Description

This parameter allows you to define the filter for your dataclass or method.

### Using a simple filter

A filter is composed of the following elements:

**{attribute} {comparator} {value}**

For example: `$filter="firstName=john"` where `firstName` is the **attribute**, `=` is the **comparator** and `john` is the **value**.

### Using a complex filter

A more complex filter is composed of the following elements, which joins two queries:

**{attribute} {comparator} {value} {AND/OR/EXCEPT} {attribute}  
{comparator} {value}**

For example: `$filter="firstName=john AND salary>20000"` where `firstName` and `salary` are attributes in the Employee dataclass.

### Using the params property

You can also use 4D's params property.

**{attribute} {comparator} {placeholder} {AND/OR/EXCEPT} {attribute}  
{comparator} {placeholder}&\$params='["{value1}","{value2}"]'"**

For example: `$filter="firstName=:1 AND salary>:2" &$params='["john",20000]'` where `firstName` and `salary` are attributes in the Employee dataclass.

For more information regarding how to query data in 4D, refer to the [dataClass.query\(\)](#) documentation.

When inserting quotes ('') or double quotes (""), you must escape them using using their character code:

- Quotes (''): \u0027
- Double quotes (""): \u0022

For example, you can write the following when passing a value with a quote when using the *params* property:

```
http://127.0.0.1:8081/rest/Person/?  
$filter="lastName=:1" &$params='["O\u0027Reilly"]'
```

If you pass the value directly, you can write the following:

```
http://127.0.0.1:8081/rest/Person/?$filter="lastName=O'Reilly"
```

## Attribute

If the attribute is in the same dataclass, you can just pass it directly (e.g., `firstName`). However, if you want to query another dataclass, you must include the relation attribute name plus the attribute name, i.e. the path (e.g., `employer.name`). The attribute name is case-sensitive (`firstName` is not equal to `FirstName`).

You can also query attributes of type Object by using dot-notation. For example, if you have an attribute whose name is "objAttribute" with the following structure:

```
{  
    prop1: "this is my first property",  
    prop2: 9181,  
    prop3: ["abc", "def", "ghi"]  
}
```

You can search in the object by writing the following:

```
GET /rest/Person/?filter="objAttribute.prop2 == 9181"
```

## Comparator

The comparator must be one of the following values:

Comparator	Description
=	equals to
!=	not equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
begin	begins with

## Examples

In the following example, we look for all employees whose last name begins with a "j":

```
GET /rest/Employee?$filter="lastName begin j"
```

In this example, we search the Employee dataclass for all employees whose salary is greater than 20,000 and who do not work for a company named Acme:

```
GET /rest/Employee?$filter="salary>20000 AND  
employer.name!=acme" &$orderby="lastName, firstName"
```

In this example, we search the Person dataclass for all the people whose number property in the anotherobj attribute of type Object is greater than 50:

```
GET /rest/Person/?filter="anotherobj.mynum > 50"
```

## \$imageformat

Defines which image format to use for retrieving images (*e.g.*, \$imageformat=png)

### Description

Define which format to use to display images. You can use one of the following formats (extensions, mime types and OsType Mac are supported):

Type	Description
"best"	Best format based on the image
".gif" or "image/gif"	GIF format
".png" or "image/png"	PNG format
".jpeg" or "image/jpeg"	JPEG format
".tiff" or "image/tiff"	TIFF format

Once you have defined the format, you must pass the image attribute to [\\$expand](#) to load the photo completely.

If there is no image to be loaded or the format doesn't allow the image to be loaded, the response will be an empty object { }.

### Example

The following example defines the image format to JPEG regardless of the actual type of the photo and passes the actual version number sent by the server:

```
GET /rest/Employee(1)/photo?  
$imageformat=.jpeg&$version=3&$expand=photo
```

## \$lock

Locks and unlocks an entity using the [pessimistic mechanism](#).

### Syntax

To lock an entity for other sessions and 4D processes:

```
/?$lock=true
```

To unlock the entity for other sessions and 4D processes:

```
/?$lock=false
```

The [lockKindText](#) property is "Locked by session".

### Description

The locks triggered by the REST API are put at the [session](#) level.

A locked entity is seen as *locked* (i.e. lock / unlock / update / delete actions are not possible) by:

- other REST sessions
- 4D processes (client/server, remote datastore, standalone) running on the REST server.

An entity locked by the REST API can only be unlocked:

- by its locker, i.e. a `/?$lock=false` in the REST session that sets `/?$lock=true`
- or if the session's [inactivity timeout](#) is reached (the session is closed).

### Response

A `?$lock` request returns a JSON object with `"result":true` if the lock operation was successful and `"result":false` if it failed.

The returned "\_\_STATUS" object has the following properties:

Property	Type	Description
success	boolean	<b>Available only in case of success:</b> true if the lock action is successful (or if the entity boolean is already locked in the current session), false otherwise (not returned in this case).
status	number	<b>Available only in case of error:</b> Error code, see below
statusText	text	Description of the error, see below
lockKind	number	Lock code
lockKindText	text	"Locked by session" if locked by a REST session, "Locked by record" if locked by a 4D process
lockInfo	object	Information about the lock origin. Returned properties depend on the lock origin (4D process or REST session). <b>Available only for a 4D process lock:</b>
task_id	number	Process ID
user_name	text	Session user name on the machine
user4d_alias	text	Name or alias of the 4D user
user4d_id	number	User id in the 4D database directory
host_name	text	Machine name
task_name	text	Process name
client_version	text	Version of the client
host	text	<b>Available only for a REST session lock:</b> URL that locked the entity (e.g. "127.0.0.1:8043")
IPAddr	text	IP address of the locker (e.g. "127.0.0.1")
recordNumber	number	Record number of the locked record
userAgent	text	userAgent of the locker (e.g. Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36")

The following values can be returned in the *status* and *statusText* properties of the *\_\_STATUS* object in case of error:

status	statusText	Comment
2	"Stamp has changed"	The internal stamp value of the entity does not match the one of the entity stored in the data (optimistic lock).
3	"Already locked"	The entity is locked by a pessimistic lock.
4	"Other error"	A serious error is a low-level database error (e.g. duplicated key), a hardware error, etc.
5	"Entity does not exist anymore"	The entity no longer exists in the data.

## Example

We lock an entity in a first browser:

```
GET /rest/Customers(1)/?$lock=true
```

**Response:**

```
{  
    "result": true,  
    "__STATUS": {  
        "success": true  
    }  
}
```

In a second browser (other session), we send the same request.

**Response:**

```
{  
    "result":false,  
    "__STATUS":{  
        "status":3,  
        "statusText":"Already Locked",  
        "lockKind":7,  
        "lockKindText":"Locked By Session",  
        "lockInfo":{  
            "host":"127.0.0.1:8043",  
            "IPAddr":"127.0.0.1",  
            "recordNumber": 7,  
            "userAgent": """Mozilla/5.0 (Macintosh; Intel Mac OS X  
10_15_3) AppleWebKit/537.36..."  
        }  
    }  
}
```

## \$method

This parameter allows you to define the operation to execute with the returned entity or entity selection.

## Available syntaxes

Syntax	Example	Description
<a href="#"><b>\$method=delete</b></a>	<pre>POST /Employee?\$filter="ID=11"&amp;\$method=delete</pre>	Deletes the current entity, entity collection, or entity selection
<a href="#"><b>\$method=entityset</b></a>	<pre>GET /People/?\$filter="ID&gt;320"&amp;\$method=entityset&amp;\$timeout=600</pre>	Creates an entity set in 4D Server's cache based on the collection of entities defined in the REST request
<a href="#"><b>\$method=release</b></a>	<pre>GET /Employee/\$entityset/&lt;entitySetID&gt;?&amp;\$method=release</pre>	Releases an existing entity set stored in 4D Server's cache
<a href="#"><b>\$method=subentityset</b></a>	<pre>GET /Company(1)/staff?&amp;\$expand=staff&amp;\$method=subentityset&amp;\$subOrderby=lastName ASC</pre>	Creates an entity set based on the collection of related entities defined in the REST request
<a href="#"><b>\$method=update</b></a>	<pre>POST /Person/?\$method=update</pre>	Updates and/or creates one or more entities

## \$method=delete

Deletes the current entity, entity collection, or entity selection (created through REST)

### Description

With `$method=delete`, you can delete an entity or an entire entity collection. You can define the collection of entities by using, for example, `$filter` or specifying one directly using `{dataClass}({key})` (e.g., `/Employee(22)`).

You can also delete the entities in an entity set, by calling

`$entityset/{entitySetID}`.

### Example

You can then write the following REST request to delete the entity whose key is 22:

```
POST /rest/Employee(22)/?$method=delete
```

You can also do a query as well using `$filter`:

```
POST /rest/Employee?$filter="ID=11"&$method=delete
```

You can also delete an entity set using `$entityset/{entitySetID}`:

```
POST /rest/Employee/$entityset/73F46BE3A0734EAA9A33CA8B14433570?  
$method=delete
```

Response:

```
{  
    "ok": true  
}
```

## \$method=entityset

Creates an entity set in 4D Server's cache based on the collection of entities defined in the REST request

### Description

When you create a collection of entities in REST, you can also create an entity set that will be saved in 4D Server's cache. The entity set will have a reference number that you can pass to `$entityset/{entitySetID}` to access it. By default, it is valid for two hours; however, you can modify that amount of time by passing a value (in seconds) to `$timeout`.

If you have used `$savedfilter` and/or `$savedorderby` (in conjunction with `$filter` and/or `$orderby`) when you created your entity set, you can recreate it with the same reference ID even if it has been removed from 4D Server's cache.

### Example

To create an entity set, which will be saved in 4D Server's cache for two hours, add `$method=entityset` at the end of your REST request:

```
GET /rest/People/?$filter="ID>320" & $method=entityset
```

You can create an entity set that will be stored in 4D Server's cache for only ten minutes by passing a new timeout to `$timeout`:

```
GET /rest/People/?$filter="ID>320" & $method=entityset & $timeout=600
```

You can also save the filter and order by, by passing true to `$savedfilter` and `$savedorderby`.

`$skip` and `$top/$limit` are not taken into consideration when saving an entity set.

After you create an entity set, the first element, `__ENTITYSET`, is added to the object returned and indicates the URI to use to access the entity set:

```
__ENTITYSET:  
"http://127.0.0.1:8081/rest/Employee/$entityset/9718A30BF61343C796345F3"
```

## \$method=release

Releases an existing entity set stored in 4D Server's cache.

### Description

You can release an entity set, which you created using `$method=entityset`, from 4D

Server's cache.

## Example

Release an existing entity set:

```
GET /rest/Employee/$entityset/4C51204DD8184B65AC7D79F09A077F24?  
$method=release
```

## Response:

If the request was successful, the following response is returned:

```
{  
    "ok": true  
}
```

If the entity set wasn't found, an error is returned:

```
{  
    "__ERROR": [  
        {  
            "message": "Error code: 1802\\nEntitySet  
\\\"4C51204DD8184B65AC7D79F09A077F24\\\" cannot be found\\ncomponent:  
'dbmg'\\ntask 22, name: 'HTTP connection handler'\\n",  
            "componentSignature": "dbmg",  
            "errCode": 1802  
        }  
    ]  
}
```

## \$method=subentityset

Creates an entity set in 4D Server's cache based on the collection of related entities defined in the REST request

## Description

`$method=subentityset` allows you to sort the data returned by the relation attribute defined in the REST request.

To sort the data, you use the `$suborderby` property. For each attribute, you specify the order as ASC (or asc) for ascending order and DESC (desc) for descending order. By default, the data is sorted in ascending order.

If you want to specify multiple attributes, you can delimit them with a comma, `,`,  
`$suborderby="lastName desc, firstName asc".`

## Example

If you want to retrieve only the related entities for a specific entity, you can make the following REST request where staff is the relation attribute in the Company dataclass linked to the Employee dataclass:

```
GET /rest/Company(1)/staff?  
$expand=staff&$method=subentityset&$suborderby=lastName ASC
```

## Response:

```

{
    "__ENTITYSET": "/rest/Employee/$entityset/FF625844008E430B9862E5FD41C741AB",
    "__entityModel": "Employee",
    "__COUNT": 2,
    "__SENT": 2,
    "__FIRST": 0,
    "__ENTITIES": [
        {
            "__KEY": "4",
            "__STAMP": 1,
            "ID": 4,
            "firstName": "Linda",
            "lastName": "Jones",
            "birthday": "1970-10-05T14:23:00Z",
            "employer": {
                "__deferred": {
                    "uri": "/rest/Company(1)",
                    "__KEY": "1"
                }
            }
        },
        {
            "__KEY": "1",
            "__STAMP": 3,
            "ID": 1,
            "firstName": "John",
            "lastName": "Smith",
            "birthday": "1985-11-01T15:23:00Z",
            "employer": {
                "__deferred": {
                    "uri": "/rest/Company(1)",
                    "__KEY": "1"
                }
            }
        }
    ]
}

```

## \$method=update

Updates and/or creates one or more entities

### Description

`$method=update` allows you to update and/or create one or more entities in a single **POST**. If you update and/or create one entity, it is done in an object with each property an attribute with its value, e.g., `{ lastName: "Smith" }`. If you update and/or create multiple entities, you must create a collection of objects.

In any cases, you must set the **POST** data in the **body** of the request.

To update an entity, you must pass the `__KEY` and `__STAMP` parameters in the object along with any modified attributes. If both of these parameters are missing, an entity will be added with the values in the object you send in the body of your **POST**.

Triggers are executed immediately when saving the entity to the server. The response contains all the data as it exists on the server.

You can also put these requests to create or update entities in a transaction by calling `$atomic/$atOnce`. If any errors occur during data validation, none of the entities are

saved. You can also use `$method=validate` to validate the entities before creating or updating them.

If a problem arises while adding or modifying an entity, an error will be returned to you with that information.

## NOTE

- **Dates** must be expressed in JS format: YYYY-MM-DDTHH:MM:SSZ (e.g., "2010-10-05T23:00:00Z"). If you have selected the Date only property for your Date attribute, the time zone and time (hour, minutes, and seconds) will be removed. In this case, you can also send the date in the format that it is returned to you dd!mm!yyyy (e.g., 05!10!2013).
- **Booleans** are either true or false.
- Uploaded files using `$upload` can be applied to an attribute of type Image or BLOB by passing the object returned in the following format `{ "ID": "D507BC03E613487E9B4C2F6A0512FE50" }`

## Example

To update a specific entity, you use the following URL:

```
POST /rest/Person/?$method=update
```

### POST data:

```
{
  KEY: "340",
  STAMP: 2,
  firstName: "Pete",
  lastName: "Miller"
}
```

The `firstName` and `lastName` attributes in the entity indicated above will be modified leaving all other attributes (except calculated ones based on these attributes) unchanged.

If you want to create an entity, you can POST the attributes using this URL:

```
POST /rest/Person/?$method=update
```

### POST data:

```
{
  firstName: "John",
  lastName: "Smith"
}
```

You can also create and update multiple entities at the same time using the same URL above by passing multiple objects in an array to the POST:

```
POST /rest/Person/?$method=update
```

### POST data:

```
[ {  
    "__KEY": "309",  
    "__STAMP": 5,  
    "ID": "309",  
    "firstName": "Penelope",  
    "lastName": "Miller"  
, {  
    "firstName": "Ann",  
    "lastName": "Jones"  
}]
```

### Response:

When you add or modify an entity, it is returned to you with the attributes that were modified. For example, if you create the new employee above, the following will be returned:

```
{  
    "__KEY": "622",  
    "__STAMP": 1,  
    "uri": "http://127.0.0.1:8081/rest/Employee(622)",  
    "__TIMESTAMP": "!!2020-04-03!!",  
    "ID": 622,  
    "firstName": "John",  
    "firstName": "Smith"  
}
```

If, for example, the stamp is not correct, the following error is returned:

```

{
    "__STATUS": {
        "status": 2,
        "statusText": "Stamp has changed",
        "success": false
    },
    "__KEY": "1",
    "__STAMP": 12,
    "__TIMESTAMP": "!!2020-03-31!!",
    "ID": 1,
    "firstname": "Denise",
    "lastname": "O'Peters",
    "isWoman": true,
    "numberOfKids": 1,
    "addressID": 1,
    "gender": true,
    "imageAtt": {
        "__deferred": {
            "uri": "/rest/Persons(1)/imageAtt?
$imageformat=best&$version=12&$expand=imageAtt",
            "image": true
        }
    },
    "extra": {
        "num": 1,
        "alpha": "I am 1"
    },
    "address": {
        "__deferred": {
            "uri": "/rest/Address(1)",
            "__KEY": "1"
        }
    },
    "__ERROR": [
        {
            "message": "Given stamp does not match current one for
record# 0 of table Persons",
            "componentSignature": "dbmrg",
            "errCode": 1263
        },
        {
            "message": "Cannot save record 0 in table Persons of
database remote_dataStore",
            "componentSignature": "dbmrg",
            "errCode": 1046
        },
        {
            "message": "The entity# 1 in the \"Persons\" dataclass
cannot be saved",
            "componentSignature": "dbmrg",
            "errCode": 1517
        }
    ]
}
}

```

## \$orderby

Sorts the data returned by the attribute and sorting order defined (e.g.,  
\$orderby="lastName desc, salary asc")

## Description

\$orderby orders the entities returned by the REST request. For each attribute, you specify the order as `ASC` (or `asc`) for ascending order and `DESC` (`desc`) for descending order. By default, the data is sorted in ascending order. If you want to specify multiple attributes, you can delimit them with a comma, e.g., \$orderby="lastName desc, firstName asc".

## Example

In this example, we retrieve entities and sort them at the same time:

```
GET /rest/Employee/?$filter="salary!=0"&$orderby="salary DESC,lastName ASC,firstName ASC"
```

The example below sorts the entity set by lastName attribute in ascending order:

```
GET /rest/Employee/$entityset/CB1BCC603DB0416D939B4ED379277F02?  
$orderby="lastName"
```

## Result:

```
{
    __entityModel: "Employee",
    __COUNT: 10,
    __SENT: 10,
    __FIRST: 0,
    __ENTITIES: [
        {
            __KEY: "1",
            __STAMP: 1,
            firstName: "John",
            lastName: "Smith",
            salary: 90000
        },
        {
            __KEY: "2",
            __STAMP: 2,
            firstName: "Susan",
            lastName: "O'Leary",
            salary: 80000
        },
        // more entities
    ]
}
```

## \$querypath

Returns the query as it was executed by 4D Server (e.g., `$querypath=true`)

### Description

`$querypath` returns the query as it was executed by 4D Server. If, for example, a part of the query passed returns no entities, the rest of the query is not executed. The query requested is optimized as you can see in this `$querypath`.

For more information about query paths, refer to [queryPlan and queryPath](#).

In the steps collection, there is an object with the following properties defining the query executed:

Property	Type	Description
description	String	Actual query executed or "AND" when there are multiple steps
time	Number	Number of milliseconds needed to execute the query
recordsfound	Number	Number of records found
steps	Collection	An collection with an object defining the subsequent step of the query path

### Example

If you passed the following query:

```
GET /rest/Employee/$filter="employer.name=acme AND  
lastName=Jones" &$querypath=true
```

And no entities were found, the following query path would be returned, if you write the following:

```
GET /rest/$querypath
```

**Response:**

```

__queryPath: {

    steps: [
        {
            description: "AND",
            time: 0,
            recordsfounds: 0,
            steps: [
                {
                    description: "Join on Table : Company : People.employer = Company.ID",
                    time: 0,
                    recordsfounds: 0,
                    steps: [
                        {
                            steps: [
                                {
                                    description: "Company.name = acme",
                                    time: 0,
                                    recordsfounds: 0
                                }
                            ]
                        }
                    ]
                }
            ]
        }
    ]
}
}

```

If, on the other hand, the first query returns more than one entity, the second one will be executed. If we execute the following query:

```
GET /rest/Employee/$filter="employer.name=a* AND lastName!=smith"&$querypath=true
```

If at least one entity was found, the following query path would be returned, if you write the following:

```
GET /rest/$querypath
```

**Response:**

```
"__queryPath": {
    "steps": [
        {
            "description": "AND",
            "time": 1,
            "recordsfounds": 4,
            "steps": [
                {
                    "description": "Join on Table : Company : Employee.employer = Company.ID",
                    "time": 1,
                    "recordsfounds": 4,
                    "steps": [
                        {
                            "steps": [
                                {
                                    "description": "Company.name LIKE a*",
                                    "time": 0,
                                    "recordsfounds": 2
                                }
                            ]
                        }
                    ]
                },
                {
                    "description": "Employee.lastName # smith",
                    "time": 0,
                    "recordsfounds": 4
                }
            ]
        }
    ]
}
```

## \$queryplan

Returns the query as it was passed to 4D Server (e.g., `$queryplan=true`)

### Description

\$queryplan returns the query plan as it was passed to 4D Server.

Property Type	Description
item	String Actual query executed
subquery Array	If there is a subquery, an additional object containing an item property (as the one above)

For more information about query plans, refer to [queryPlan and queryPath](#).

### Example

If you pass the following query:

```
GET /rest/People/$filter="employer.name=acme AND
lastName=Jones"&$queryplan=true
```

### Response:

```
__queryPlan: {
    And: [
        {
            item: "Join on Table : Company : People.employer =
Company.ID",
            subquery: [
                {
                    item: "Company.name = acme"
                }
            ]
        },
        {
            item: "People.lastName = Jones"
        }
    ]
}
```

## \$savedfilter

Saves the filter defined by \$filter when creating an entity set (e.g., `$savedfilter="$filter"`)

### Description

When you create an entity set, you can save the filter that you used to create it as a measure of security. If the entity set that you created is removed from 4D Server's cache (due to the timeout, the server's need for space, or your removing it by calling [\\$method=release](#)).

You use `$savedfilter` to save the filter you defined when creating your entity set and then pass `$savedfilter` along with your call to retrieve the entity set each time.

If the entity set is no longer in 4D Server's cache, it will be recreated with a new default timeout of 10 minutes. The entity set will be refreshed (certain entities might be included while others might be removed) since the last time it was created, if it no longer existed before recreating it.

If you have used both `$savedfilter` and `$savedorderby` in your call when creating an entity set and then you omit one of them, the new entity set, which will have the same reference number, will reflect that.

### Example

In our example, we first call `'$savedfilter'` with the initial call to create an entity set as shown below:

```
GET /rest/People/?  
$filter="employer.name=Apple"&$savedfilter="employer.name=Apple"&$meth  
od=entityset
```

Then, when you access your entity set, you write the following to ensure that the entity set is always valid:

```
GET /rest/People/$entityset/AEA452C2668B4F6E98B6FD2A1ED4A5A8?  
$savedfilter="employer.name=Apple"
```

## \$savedorderby

Saves the order by defined by `$orderby` when creating an entity set (e.g., `$savedorderby="{orderby}"`)

### Description

When you create an entity set, you can save the sort order along with the filter that you used to create it as a measure of security. If the entity set that you created is removed from 4D Server's cache (due to the timeout, the server's need for space, or your removing it by calling [\\$method=release](#)).

You use `$savedorderby` to save the order you defined when creating your entity set, you then pass `$savedorderby` along with your call to retrieve the entity set each time.

If the entity set is no longer in 4D Server's cache, it will be recreated with a new default timeout of 10 minutes. If you have used both [\\$savedfilter](#) and `$savedorderby` in your call when creating an entity set and then you omit one of them, the new entity set, having the same reference number, will reflect that.

### Example

You first call `$savedorderby` with the initial call to create an entity set:

```
GET /rest/People/?
$filter="lastName!=''"&$savedfilter="lastName!=''"&$orderby="salary"&$
savedorderby="salary"&$method=entityset
```

Then, when you access your entity set, you write the following (using both `$savedfilter` and `$savedorderby`) to ensure that the filter and its sort order always exists:

```
GET /rest/People/$entityset/AEA452C2668B4F6E98B6FD2A1ED4A5A8?
$filter="lastName!=''"&$savedorderby="salary"
```

## \$skip

Starts the entity defined by this number in the collection (e.g., `$skip=10`)

### Description

`$skip` defines which entity in the collection to start with. By default, the collection sent starts with the first entity. To start with the 10th entity in the collection, pass 10.

`$skip` is generally used in conjunction with [`\$top/\$limit`](#) to navigate through an entity collection.

### Example

In the following example, we go to the 20th entity in our entity set:

```
GET /rest/Employee/$entityset/CB1BCC603DB0416D939B4ED379277F02?  
$skip=20
```

## \$timeout

Defines the number of seconds to save an entity set in 4D Server's cache (e.g., \$timeout=1800)

### Description

To define a timeout for an entity set that you create using [\\$method=entityset](#), pass the number of seconds to [\\$timeout](#). For example, if you want to set the timeout to 20 minutes, pass 1200. By default, the timeout is two (2) hours.

Once the timeout has been defined, each time an entity set is called upon (by using [\\$method=entityset](#)), the timeout is recalculated based on the current time and the timeout.

If an entity set is removed and then recreated using [\\$method=entityset](#) along with [\\$savedfilter](#), the new default timeout is 10 minutes regardless of the timeout you defined when calling [\\$timeout](#).

### Example

In our entity set that we're creating, we define the timeout to 20 minutes:

```
GET /rest/Employee/?  
$filter="salary!=0"&$method=entityset&$timeout=1200
```

# Page Not Found

We could not find what you were looking for.

Please contact the owner of the site that linked you to the original URL and let them know their link is broken.

## \$version

Image version number

### Description

`$version` is the image's version number returned by the server. The version number, which is sent by the server, works around the browser's cache so that you are sure to retrieve the correct image.

The value of the image's version parameter is modified by the server.

### Example

The following example defines the image format to JPEG regardless of the actual type of the photo and passes the actual version number sent by the server:

```
GET /rest/Employee(1)/photo?$imageformat=jpeg&$version=3&$expand=photo
```

# Desktop Applications

Guides for developing Desktop applications with 4D

## [Client/Server](#)

[4D Desktop applications can be used in a Client/Server configuration, either as merged or separate applications.](#)

## [Access Rights](#)

[2 items](#)

## [Forms](#)

[7 items](#)

## [Menus](#)

[4 items](#)

## [User Settings](#)

[4D provides two modes of operation for project Settings:](#)

## [Build Application](#)

[4D includes an application builder to create a project package \(final build\). This build...](#)

# Client/Server Management

4D Desktop applications can be used in a Client/Server configuration, either as merged client/server applications or as remote projects.

- **merged client/server applications** are generated by the [Build Application manager](#). They are used for application deployments.
- **remote projects** are [.4DProject](#) files opened by 4D Server and accessed with 4D in remote mode. The server sends a .4dz version of the project ([compressed format](#)) to the remote 4D, thus structure files are read-only. This configuration is usually used for application testing.

Connecting to a remote project from the **same machine as 4D Server** allows modifying the project files. This [specific feature](#) allows to develop a client/server application in the same context as the deployment context.

## Opening a merged client/server application

A merged client/server application is customized and its starting is simplified:

- To launch the server portion, the user simply double-clicks on the server application. The project file does not need to be selected.
- To launch the client portion, the user simply double-clicks the client application, which connects directly to the server application.

These principles are detailed in the [Build Application](#) page.

## Opening a remote project

The first time you connect to a 4D Server project via a remote 4D, you will usually use the standard connection dialog. Thereafter, you will be able to connect directly using the **Open Recent Projects** menu or a 4DLink shortcut file.

To connect remotely to a 4D Server project:

1. Do one of the following:
  - Select **Connect to 4D Server** in the Welcome Wizard dialog
  - Select **Open/Remote Project...** from the **File** menu or the **Open** toolbar button.

The 4D Server connection dialog appears. This dialog has three tabs: **Recent**, **Available**, and **Custom**.

If 4D Server is connected to the same network as the remote 4D, select **Available**. 4D Server includes a built-in TCP/IP broadcasting system that, by default, publishes the name of the 4D Server projects available over the network. The list is sorted by order of appearance and updated dynamically.

To connect to a server from the list, double-click on its name or select it and click the **OK** button.

A circumflex accent (^) is placed before the name of projects published with

the encryption option enabled.

If the published project is not displayed in the **Available** list, select **Custom**. The Custom page allows you to connect to a published server on the network using its network address and assigning it a customized name.

- **Project name:** Defines the local name of the 4D Server project. This name will be used in the **Recent** page when referring to the project.
- **Network address:** The IP address of the machine where the 4D Server was launched.
  - If two servers are executed simultaneously on the same machine, the IP address must be followed by a colon and port number, for example:  
192.168.92.104:19814.
  - By default, the publishing port of a 4D Server is 19813. This number can be modified in the Project settings.

The **Activate development mode** option opens the remote connection in a special read/write mode and requires to access the project folder from the remote 4D (compatibility option).

Once this page assigns a server, clicking the **OK** button will allow you to connect to the server.

Once a connection to the server has been established, the remote project will be listed on the **Recent** tab.

### Updating project files on the server

4D Server automatically creates and sends the remote machines a [.4dz version](#) of the *.4DProject* project file (not compressed) in interpreted mode.

- An updated .4dz version of the project is automatically produced when necessary, i.e. when the project has been modified and reloaded by 4D Server. The project is reloaded:
  - automatically, when the 4D Server application window comes to the front of the OS or when the 4D application on the same machine saves a modification (see below).
  - when the `RELOAD PROJECT` command is executed. Calling this command is necessary for example when you have pulled a new version of the project from the source control platform.

### Updating project files on remote machines

When an updated .4dz version of the project has been produced on 4D Server, connected remote 4D machines must log out and reconnect to 4D Server in order to benefit from the updated version.

## Using 4D and 4D Server on the same machine

When 4D connects to a 4D Server on the same machine, the application behaves as 4D in single user mode and the design environment allows you to edit project files. This feature allows you to develop a client/server application in the same context as the deployment context.

When 4D connects to a 4D Server on the same machine, the **development mode** is automatically activated, whatever the [opening option](#) status.

Each time 4D performs a **Save all** action from the design environment (explicitly from **File** menu or implicitly by switching to application mode for example), 4D Server synchronously reloads project files. 4D waits for 4D Server to finish reloading the project files before it continues.

However, you need to pay attention to the following behavior differences compared to [standard project architecture](#):

- the userPreferences.{username} folder used by 4D is not the same folder used by 4D Server in the project folder. Instead, it is a dedicated folder, named "userPreferences", stored in the project system folder (i.e., the same location as when opening a .4dz project).
- the folder used by 4D for derived data is not the folder named "DerivedData" in the project folder. Instead it is a dedicated folder named "DerivedDataRemote" located in the project system folder.
- the catalog.4DCatalog file is not edited by 4D but by 4D Server. Catalog information is synchronised using client/server requests
- the directory.json file is not edited by 4D but by 4D Server. Directory information is synchronised using client/server requests
- 4D uses its own internal components and plug-ins instead of those in 4D Server.

It is not recommended to install plug-ins or components at the 4D or 4D Server application level.

# Access Rights

Access control and user privileges for desktop applications.

## Access Control overview

If more than one person uses an application, which is usually the case in client-server environments, access control is required to manage who can do what in the application.

## Managing 4D users and groups

In multi-user applications, 4D provides users with certain standard access privileges and allows you to define your own custom privileges.

## Access Control overview

If more than one person uses an application, which is usually the case in client-server architecture or Web interfaces, you need to control access or provide different features according to the connected users. It is also essential to provide security for sensitive data, even in single-user applications.

4D access control strategy depends on your deployment configuration:

- in multi-user applications, you can rely on 4D users and groups,
- in single-user applications, user access is controlled through the system session, using commands such as [Current system user](#).

For an overview of 4D's security features, see the [4D Security guide](#).

## Access control in multi-user applications

Multi-user applications are deployed with 4D Server. They include client-server, Web, or REST applications.

In multi-user applications, access control is done through [4D users and groups](#). You create users, assign passwords, create access groups that have different levels of privileges in the application.

You initiate the 4D password access control system with 4D Server by [assigning a password to the Designer user](#). Until you give the Designer a password, all application access are done with the Designer's access rights, even if you have [set up users and groups](#) (when the application opens, no ID is required). Any part of the application can be opened.

When a password is assigned to the Designer, all the access privileges take effect. In order to connect to the application or to a [server with protected access](#), remote users must enter a login/password.

To disable the password access system, you just need to remove the Designer password.

## Access control in single-user applications

Single-user applications are desktop applications, deployed with 4D or merged with 4D Volume License. In single-user applications all users opening the application are [Designers](#), they have all privileges and their name is "Designer". Access control is not based upon 4D users and groups, but upon [user sessions](#).

### User identification

To identify the current user in a 4D single-user application, you can rely on the [Current system user](#) command, which returns the user who opened the system session. Thus user authentication is delegated to the OS level.

You can then allow or deny access within your application by using code such as:

```
If(Current system user = $user) //you can store users in a database  
table  
    // give access to some features  
End if
```

If you want to use the system user name in 4D instead of "Designer" (e.g. in log files), you can call the [SET\\_USER\\_ALIAS](#) command, for example:

```
SET USER ALIAS (Current system user)
```

## Protecting access

### Privileges

On a machine that is shared by several users, you can install the 4D application in a folder and give appropriate user access privileges to the folder at the OS level.

### Encrypting data

If you want to protect access to the application data, we recommend to [encrypt data](#) and provide the encryption key to the authorized user(s).

## Managing 4D users and groups

In multi-user applications, 4D provides users with certain standard access privileges and certain powers. Once a users and groups system has been initiated, these standard privileges take effect.

## Users and groups in projects

In project applications (.4DProject or .4dz files), 4D users and groups can be configured in both single-user and multi-user environments. However, **access control** is only effective with 4D Server. The following table lists the main users and groups features and their availability:

	4D (single-user)	4D Server
Adding/editing users and groups	yes	yes
Assigning user/group access to servers	yes	yes
User identification	no (all users are Designer)	yes
Access control once the Designer has been assigned a password	no (all access are Designer)	yes

For information about user identification and access control in single-user deployments, see [this paragraph](#).

## Designer and Administrator

The most powerful user is named **Designer**. No aspect of the application is closed to the Designer. The Designer can:

- access all application servers without restriction,
- create users and groups,
- assign access privileges to groups,
- access the Design environment. In single-user environment, Designer access rights are always used. In client/server environment, assigning a password to the Designer activates the display of the 4D user login dialog. Access to Design environment is read-only.

After the Designer, the next most powerful user is the **Administrator**, who is usually given the tasks of managing the access system and administration features.

The Administrator can:

- create users and groups,
- access the 4D Server Administration window and monitor
- access the MSC window to monitor backup, restore, or server.

The Administrator cannot:

- edit the Designer user
- by default, access to protected parts of the application. In particular, the Administrator cannot access to the Design mode if it is restricted. The Administrator must be part of one or more groups to have access privileges in the application. The Administrator is placed in every new group, but you can remove the Administrator's name from any group.

Both the Designer and Administrator are available by default in all applications. In the [user management dialog box](#), the icons of the Designer and Administrator are displayed in red and green respectively:

- Designer icon: 
- Administrator icon: 

You can rename the Designer and Administrator users. In the language, the Designer ID is always 1 and the Administrator ID is always 2.

The Designer and Administrator can each create up to 16,000 groups and 16,000 users.

## Users editor

The editor for users is located in the Toolbox of 4D.

Users and groups editor can be displayed at runtime using the [EDIT ACCESS](#) command. The whole users and groups configuration can also be edited during application execution using 4D language commands of the `Users` and `Groups` theme.

### Adding and modifying users

You use the users editor to create user accounts, set their properties and assign them to various groups.

To add a user from the Toolbox :

1. Select **Tool Box > Users** from the **Design** menu or click on the **Tool Box** button of the 4D toolbar. 4D displays the users editor.

The list of users displays all the users, including the [Designer and the Administrator](#).

2. Click on the  button located below the list of users. OR Right-click in the list of users and choose **Add** or **Duplicate** in the context menu.

The **Duplicate** command can be used to create several users having the same characteristics quickly.

4D adds a new user to the list, named "New userX" by default.

3. Enter the user name. This name will be used by the user to open the application. You can rename a user at any time using the **Rename** command of the context menu, or by using the Alt+click (Windows) or Option+click (macOS) shortcuts, or by clicking twice on the name you want to change.
4. To enter a password for the user, click the **Edit...** button in the user properties area and enter the password twice in the dialog box. You can use up to 15 alphanumeric characters for a password. The password editor is case sensitive.

Users can change their password at any time according to the options in the "Security" page of the structure settings, or using the [CHANGE PASSWORD](#) command.

5. Set the group(s) to which the user belongs using the "Member of Groups" table. You can add or remove the selected user to/from a group by checking the corresponding option in the Member column.

The membership of users to different groups can also be set by group on the [Groups page](#).

## Deleting a user

To delete a user, select it then click the deletion button or use the **Delete** command of the context menu. 

Deleted user names no longer appear in the Users editor. Note that the IDs for deleted users are reassigned when new user accounts are created.

## User properties

- **User Kind:** The User Kind field contains "Designer", "Administrator", or (for all other users) "User".
- **Startup Method:** Name of an associated method that will be automatically executed when the user opens the application (optional). This method can be used for example to load the user preferences.

## Groups editor

The editor for groups is located in the Toolbox of 4D.

### Configuring groups

You use the groups editor to set the elements that each group contains (users and/or other groups) and to distribute access to plug-ins.

Keep in mind that once a group has been created, it cannot be deleted. If you want to deactivate a group, you just need to remove any users it contains.

To create a group:

1. Select **Tool Box > Groups** in the **Design** menu or click on the **Tool Box** button of the 4D toolbar then on the **Groups** button. 4D displays the groups editor window. The list of groups displays all the groups of the application project.
2. Click on the  button located below the list of groups.  
OR  
Right-click in the list of groups and choose the **Add** or **Duplicate** command in the context menu.

The Duplicate command can be used to create several groups having the same characteristics quickly.

4D adds a new group to the list, named "New groupX" by default.

3. Enter the name of the new group. The group name can be up to 15 characters long. You can rename a group at any time using the **Rename** command of the context menu, or by using the Alt+click (Windows) or Option+click (macOS) shortcuts, or by clicking twice on the name you want to change.

### Placing users or groups into groups

You can place any user or group into a group, and you can also place the group itself into several other groups. It is not mandatory to place a user in a group.

To place a user or group in a group, you simply need to check the "Member" option for each user or group in the member attribution area:

	User / Group	Member
	Administrator	<input checked="" type="checkbox"/>
	Designer	<input type="checkbox"/>
	New user	<input type="checkbox"/>
	Paul	<input type="checkbox"/>
	Peter	<input checked="" type="checkbox"/>
	Sarah	<input type="checkbox"/>
	Finances	<input checked="" type="checkbox"/>
	General Management	<input checked="" type="checkbox"/>
	Devs	<input type="checkbox"/>
	Admins	<input type="checkbox"/>

If you check the name of a user, this user is added to the group. If you check the name of a group, all the users of the group are added to the new group. The affiliated user or group will then have the same access privileges as those assigned to the new group.

Placing groups into other groups lets you create a user hierarchy. The users of a group placed in another group will have the access privileges of both groups. See "[An access hierarchy scheme](#)" below.

To remove a user or group from another group, you just need to deselect the corresponding option in the member attribution area.

### Assigning a group to a plug-in or to a server

You can assign a group privileges to any plug-ins installed in the project. This includes all the 4D plug-ins and any third-party plug-ins.

Distributing access to the plug-ins lets you control the use of the licenses you possess for these plug-ins. Any users that do not belong to the access group of a plug-in cannot load this plug-in.

Used licenses remain attached to 4D user accounts in the group for the whole 4D session.

The "Plug-in" area on the Groups page of the tool box lists all the plug-ins loaded by the 4D application. To give a group access to a plug-in, you simply need to check the corresponding option.

	Plug-in	Access
	4D Client Web Server	<input type="checkbox"/>
	4D Client SOAP Server	<input type="checkbox"/>
	4D Write PRO	<input checked="" type="checkbox"/>
	4D View PRO	<input type="checkbox"/>

The **4D Client Web Server** and **4D Client SOAP Server** items lets you control the possibility of Web and SOAP (Web Services) publication for each 4D in remote mode. These licenses are considered as plug-in licenses by 4D Server. Therefore, in the same way as for plug-ins, you can restrict the right to use these licenses to a specific group of users.

### An access hierarchy scheme

The best way to ensure the security of your application and provide users with different levels of access is to use an access hierarchy scheme. Users can be assigned to appropriate groups and groups can be nested to create a hierarchy of access rights.

This section discusses several approaches to such a scheme.

In this example, a user is assigned to one of three groups depending on their level of responsibility. Users assigned to the Accounting group are responsible for data entry. Users assigned to the Finances group are responsible for maintaining the data, including updating records and deleting outdated records. Users assigned to the General Management group are responsible for analyzing the data, including performing searches and printing analytical reports.

The groups are then nested so that privileges are correctly distributed to the users of each group.

- The General Management group contains only “high-level” users.
- The Finances group contains data maintenance users as well as General Management users, thus the users in General Management have the privileges of the Finances group as well.
- The Accounting group contains data entry users as well as Finances group users, so the users who belong to the Finances group and the General Management group enjoy the privileges of the Accounting group as well.

You can decide which access privileges to assign to each group based on the level of responsibility of the users it includes.

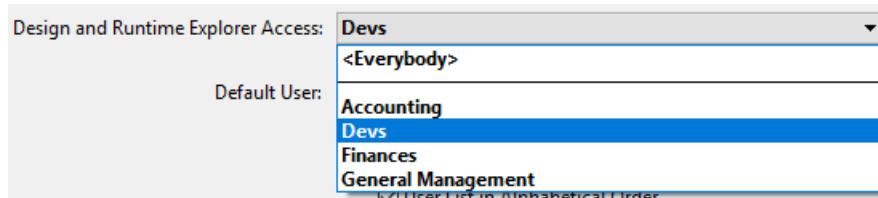
Such a hierarchical system makes it easy to remember to which group a new user should be assigned. You only have to assign each user to one group and use the hierarchy of groups to determine access.

## Assigning group access

Groups are assigned access privileges to specific parts or features of the application:

- Design and Runtime Explorer access,
- HTTP server,
- REST server,
- SQL server.

These accesses are defined in the Settings dialog. The following example shows Design and Runtime explorer access rights being assigned to the "Devs" group:



You also use groups to [distribute available licenses](#). This distribution is defined in the Groups editor.

## Directory.json file

Users, groups, as well as their access rights are stored in a specific project file named **directory.json**.

This file can be stored at the following locations, depending on your needs:

- If you want to use the same directory for all data files (or if you use a single data file), store the **directory.json** file in the user settings folder, i.e. in the "Settings" folder at the [same level as the "Project" folder](#) (default location).
- If you want to use a specific directory file per data file, store the **directory.json** file in the data settings folder, i.e. in the ["Settings" folder of the "Data" folder](#). If a **directory.json** file is present at this location, it takes priority over the file in the user settings folder. This custom/local Users and Groups configuration will left untouched by an application upgrade.

To allow for safe changes of passwords and group memberships in a deployed environment, you can include your **directory.json** file in the server application during the build, using the [corresponding build application option](#).

## Forms

Forms provide the interface through which information is entered, modified, and printed in a desktop application. Users interact with the data in a database using forms and print reports using forms. Forms can be used to create custom dialog boxes, palettes, or any featured custom window.

Forms can also contain other forms through the following features:

- [subform objects](#)
- [inherited forms](#)

## Creating forms

You can add or modify 4D forms using the following elements:

- **4D Developer interface:** Create new forms from the **File** menu or the **Explorer** window.
- **Form Editor:** Modify your forms using the [Form Editor](#).
- **JSON code:** Create and design your forms using JSON and save the form files at the [appropriate location](#). Example:

```

{
    "windowTitle": "Hello World",
    "windowMinWidth": 220,
    "windowMinHeight": 80,
    "method": "HWexample",
    "pages": [
        null,
        {
            "objects": {
                "text": {
                    "type": "text",
                    "text": "Hello World!",
                    "textAlign": "center",
                    "left": 50,
                    "top": 120,
                    "width": 120,
                    "height": 80
                },
                "image": {
                    "type": "picture",
                    "pictureFormat": "scaled",
                    "picture": "/RESOURCES/Images/HW.png",
                    "alignment": "center",
                    "left": 70,
                    "top": 20,
                    "width": 75,
                    "height": 75
                },
                "button": {
                    "type": "button",
                    "text": "OK",
                    "action": "Cancel",
                    "left": 60,
                    "top": 160,
                    "width": 100,
                    "height": 20
                }
            }
        }
    ]
}

```

## Project form and Table form

There are two categories of forms:

- **Project forms** - Independent forms that are not attached to any table. They are intended more particularly for creating interface dialog boxes as well as components. Project forms can be used to create interfaces that easily comply with OS standards.
- **Table forms** - Attached to specific tables and thus benefit from automatic functions useful for developing applications based on databases. Typically, a table has separate input and output forms.

Typically, you select the form category when you create the form, but you can change it afterwards.

## Form pages

Each form has is made of at least two pages:

- a page 1: a main page, displayed by default
- a page 0: a background page, whose contents is displayed on every other page.

You can create multiple pages for an input form. If you have more fields or variables than will fit on one screen, you may want to create additional pages to display them. Multiple pages allow you to do the following:

- Place the most important information on the first page and less important information on other pages.
- Organize each topic on its own page.
- Reduce or eliminate scrolling during data entry by setting the [entry order](#).
- Provide space around the form elements for an attractive screen design.

Multiple pages are a convenience used for input forms only. They are not for printed output. When a multi-page form is printed, only the first page is printed.

There are no restrictions on the number of pages a form can have. The same field can appear any number of times in a form and on as many pages as you want. However, the more pages you have in a form, the longer it will take to display it.

A multi-page form has both a background page and several display pages. Objects that are placed on the background page may be visible on all display pages, but can be selected and edited only on the background page. In multi-page forms, you should put your button palette on the background page. You also need to include one or more objects on the background page that provide page navigation tools for the user.

## Inherited Forms

4D forms can use and be used as "inherited forms," meaning that all of the objects from *Form A* can be used in *Form B*. In this case, *Form B* "inherits" the objects from *Form A*.

References to an inherited form are always active: if an element of an inherited form is modified (button styles, for example), all forms using this element will automatically be modified.

All forms (table forms and project forms) can be designated as an inherited form. However, the elements they contain must be compatible with use in different database tables.

When a form is executed, the objects are loaded and combined in the following order:

1. Page zero of the inherited form
2. Page 1 of the inherited form
3. Page zero of the open form
4. Current page of the open form.

This order determines the default [entry order](#) of objects in the form.

Only pages 0 and 1 of an inherited form can appear in other forms.

The properties and method of a form are not considered when that form is used as an inherited form. On the other hand, the methods of objects that it contains are called.

To define an inherited form, the [Inherited Form Name](#) and [Inherited Form Table](#) (for table form) properties must be defined in the form that will inherit something from another form.

A form can inherit from a project form, by setting the [Inherited Form Table](#) property

to `\<None>` in the Property List (or " " in JSON).

To stop inheriting a form, select `\<None>` in the Property List (or " " in JSON) for the [Inherited Form Name](#) property.

It is possible to define an inherited form in a form that will eventually be used as an inherited form for a third form. The combining of objects takes place in a recursive manner. 4D detects recursive loops (for example, if form [table1]form1 is defined as the inherited form of [table1]form1, in other words, itself) and interrupts the form chain.

## Supported Properties

[Associated Menu Bar](#) - [Fixed Height](#) - [Fixed Width](#) - [Form Break](#) - [Form Detail](#) - [Form Footer](#) - [Form Header](#) - [Form Name](#) - [Form Type](#) - [Inherited Form Name](#) - [Inherited Form Table](#) - [Maximum Height](#) - [Maximum Width](#) - [Method](#) - [Minimum Height](#) - [Minimum Width](#) - [Pages](#) - [Print Settings](#) - [Published as Subform](#) - [Save Geometry](#) - [Window Title](#)

## Style sheets

A style sheet groups together a combination of attributes for form objects — from text attributes to nearly any available object attribute.

In addition to harmonizing an application's interface, style sheets provide three major advantages:

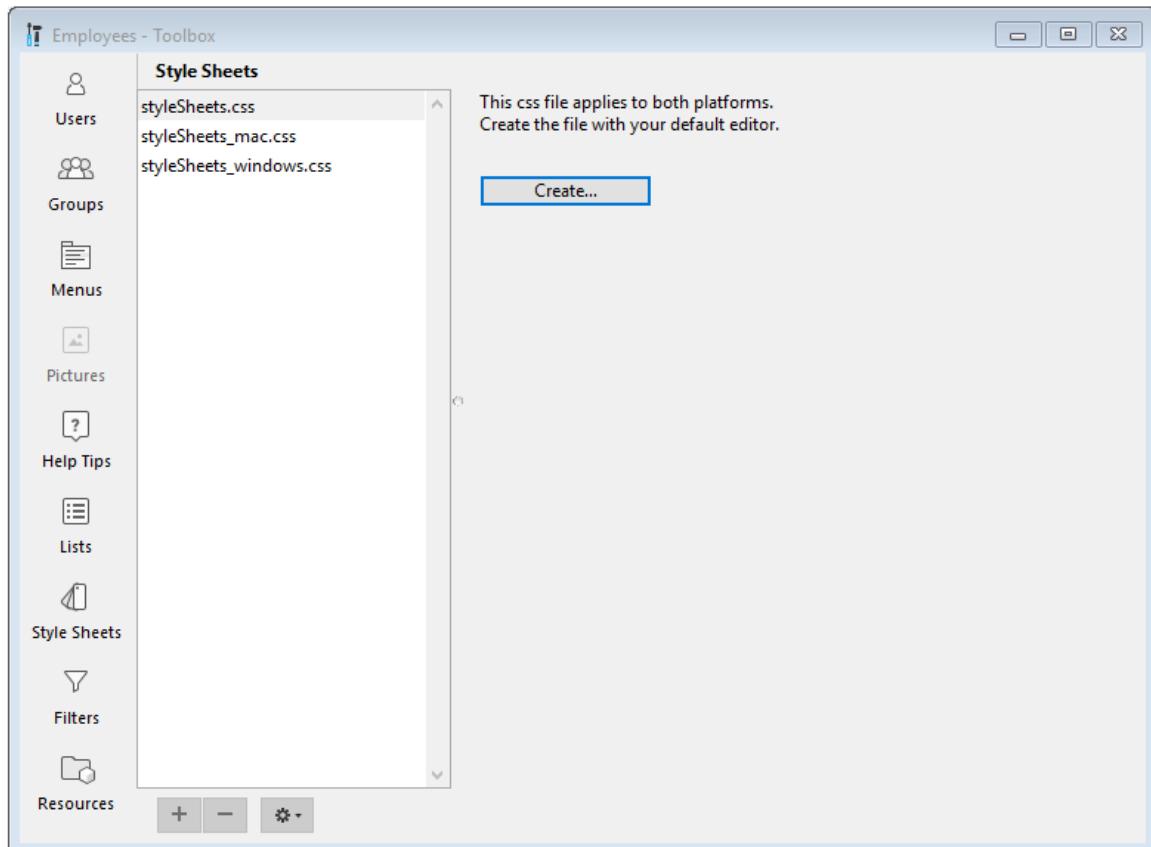
- Saves time during development: Each object has specific group of settings within a single operation.
- Facilitates maintenance: Style sheets modify the appearance of any objects that uses them, so changing the font size in a style sheet will change the font size for all of the objects that use this same style sheet.
- Controls multi-platform development: You can have a style sheets that apply to both macOS and Windows platforms, only macOS, or only Windows. When a style sheet is applied, 4D automatically uses the appropriate style sheet.

## Creating or Editing Style Sheets

You can create style sheets using your preferred text editor and saving the file with a ".css" extension in the project's "/SOURCES" folder.

The 4D Tool Box provides a **Style Sheets** page as a shortcut option to create and edit one of three platform-specific named style sheets.

1. Open the **Style Sheets** page by choosing the **Tool Box > Style Sheet** from the Design menu or click on the **Tool Box** icon in the Form Editor toolbar.



2. Select the type of style sheet to create and click on the **Create** or **Edit** button:

**Create...**

3. The style sheet will open in your default text editor.

## Style Sheet Files

4D accepts three, specific style sheet files:

Style Sheet	Platform
styleSheets.css	Default global style sheet for both macOS and Windows
styleSheets_mac.css	For defining macOS only specific attribute styles
styleSheets_windows.css	For defining Windows only specific attribute styles

These files are stored in the project's "/SOURCES" folder. They can also be accessed directly via the [CSS Preview](#) in the Form editor toolbar.

## Style Sheet Architecture

While adapted to meet the specific needs of 4D forms, style sheets for application projects generally follow CSS2 syntax and grammar.

Every style rule in a style sheet contains two parts:

- a *Selector* - A selector defines where to apply the style. 4D supports "object type", "object name", "class", "all objects", as well as "attribute value" selectors.
- a *Declaration* - The declaration defines the actual style to apply. Multiple declaration lines can be grouped together to form a declaration block. Each line in a CSS declaration block must end with a semicolon, and the entire block must be surrounded by curly braces.

## Style Sheet Selectors

### Object Type

Corresponding to the CSS element selector, the object type defines the type of object to style.

Specify the object type, then in curly braces, declare the style(s) to apply.

The object type corresponds to the JSON [type](#) property of form objects.

In the following example, all objects of the *button* type will display text in the Helvetica Neue font, with a size of 20 pixels:

```
button {  
    font-family: Helvetica Neue;  
    font-size: 20px;  
}
```

To apply the same style to multiple types of objects, specify the object types separated by a "," then in curly braces, declare the style(s) to apply:

```
text, input {  
    text-align: left;  
    stroke: grey;  
}
```

### Object Name

Corresponding to the CSS **ID selector**, the object name defines a specific object to style since the object's name is unique within the form.

Designate the object with a "#" character before the object's name, then in curly braces, declare the style(s) to apply.

In the following example, the text of the object with the name "okButton" will be displayed in Helvetica Neue font, with a size of 20 pixels:

```
#okButton {  
    font-family: Helvetica Neue;  
    font-size: 20px;  
}
```

## Class

Corresponding to the CSS **class selector**, the class defines the style for all form objects with the `class` attribute.

You can specify the classes to use with a "." character followed by the name of the class, and in curly braces, declare the style(s) to apply.

In the following example, the text of all objects with the `okButtons` class will be displayed in Helvetica Neue font, with a size of 20 pixels, aligned in the center:

```
.okButtons {  
    font-family: Helvetica Neue;  
    font-size: 20px;  
    text-align: center;  
}
```

To designate that a style should be applied only to objects of a distinct type, specify the type followed by "." and the name of the class, then in curly braces, declare the style(s) to apply.

```
text.center {  
    text-align: center;  
    stroke: red;  
}
```

In the 4D form description, you associate a class name to an object using the `class` attribute. This attribute contains one or several class names, separated by a space character:

```
class: "okButtons important"
```

## All Objects

Corresponding to the CSS **universal selector**, the "\*" character indicates that the following style will be applied to all objects on the form.

Designate that a style should apply to all form objects with the "\*" character, then in curly braces, declare the style(s) to apply.

In the following example, all objects will have a gray fill:

```
* {  
    fill: gray;  
}
```

## Specific Attribute

Corresponding to the CSS **attribute selectors**, styles can be applied to all form objects with a specific attribute.

Specify the attribute within brackets, then in curly braces, declare the style(s) to apply.

## Supported syntaxes

Syntax	Description
[attribute]	matches objects with the <code>attribute</code>
[attribute="value"]	matches objects with the <code>attribute</code> value containing exactly the specified "value"
[attribute~="value"]	matches objects with the <code>attribute</code> value containing the "value" among a space-separated list of words
[attribute ="value"]	matches objects with an <code>attribute</code> whose value starts with "value"

## Examples

All objects with the `borderStyle` attribute will have purple lines:

```
[borderStyle]
{
    stroke: purple;
}
```

All objects of the text type with a text attribute whose value is "Hello" will have blue letters:

```
text[text=Hello]
{
    stroke: blue;
}
```

All objects with a text attribute whose value contains "Hello" will have blue lines:

```
[text~=Hello]
{
    stroke: blue;
}
```

All objects of the text type with a text attribute whose value starts with "Hello" will have yellow letters:

```
text[text|=Hello]
{
    stroke: yellow;
}
```

## Style Sheet Declarations

### Media Queries

Media queries are used to apply color schemes to an application.

A media query is composed of a media feature and a value (e.g., `\<media feature>:\<value>` ).

Available media features:

- `prefers-color-scheme`

Available media feature expressions:

- **light**  
For using a light scheme
- **dark**  
For using a dark scheme

Color schemes are only supported on macOS.

## Example

This CSS defines a color combination for text and text background in the light scheme (default) and another combination when the dark scheme is selected:

```
@media (prefers-color-scheme: light) {
  .textScheme {
    fill: LightGrey;
    stroke: Black;
  }
}

@media (prefers-color-scheme: dark) {
  .textScheme {
    fill: DarkSlateGray;
    stroke: LightGrey;
  }
}
```

## Object Attributes

The majority of form object attributes can be defined within a style sheet, except the following attributes:

- `method`
- `type`
- `class`
- `event`
- `choiceList`, `excludedList`, `labels`, `list`, `requiredList` (list type)

Form object attributes can be declared with their [JSON name](#) as CSS attributes (not including object types, methods, events, and lists).

## Attribute Mapping

The attributes listed below are able to accept either the 4D name or the CSS name.

4D	CSS
borderStyle	border-style
fill	background-color
fontFamily	font-family
fontSize	font-size
fontStyle	font-style
fontWeight	font-weight
stroke	color
textAlign	text-align
textDecoration	text-decoration
verticalAlign	vertical-align

4D-specific values (e.g., `sunken`) are not supported when using CSS attribute names.

## Specific Attribute Values

- For `icon`, `picture`, and `customBackgroundPicture` attributes that support a path to an image, the syntax is:

```
icon: url("/RESOURCES/Images/Buttons/edit.png"); /* absolute path */
icon: url("edit.png"); /* relative path to the form file */
```

- For `fill`, `stroke`, `alternateFill`, `horizontalLineStroke` and `verticalLineStroke`, three syntaxes are supported:

- CSS color name: `fill: red;`
- Hexa value: `fill: #FF0000;`
- the `rgb()` function: `fill:rgb(255,0,0)`

- If a string uses forbidden characters in CSS, you can surround the string with simple or double quotes. For example:

- a xliff reference: `tooltip: ":xiff:CommonMenuFile";`
- a datasource with a field expression: `dataSource: "[Table_1:1]ID:1";`

## Priority Order

4D projects prioritizes conflicting style definitions first by the form definition, then by the style sheets.

### JSON vs Style Sheet

If an attribute is defined in the JSON form description and a style sheet, 4D will use the value in the JSON file.

To override this behavior, the style value must be followed with an `!important` declaration.

#### Example 1:

##### JSON form description Style Sheet 4D displays

```
"text": "Button",    text: Edit; "Button"
```

#### Example 2:

## JSON form description

```
"text": "Button",    text: Edit !important; "Edit"
```

## Style Sheet

## 4D displays

### Multiple Style Sheets

At runtime, 4D automatically prioritizes style sheets in the following order:

1. The 4D form will first load the default CSS file `/SOURCES/styleSheets.css`.
2. It will then load the CSS file for the current platform  
`/SOURCES/styleSheets_mac.css` or `/SOURCES/styleSheets_windows.css`.
3. If it exists, it will then load a specific CSS file defined in the JSON form:

- a file for both platforms:

```
"css": "<path>"
```

- or a list of files for both platforms:

```
"css": [  
    "<path1>",  
    "<path2>"  
,
```

- or a list of files per platform:

```
"css": [  
    {"path": "<path>", "media": "mac"},  
    {"path": "<path>", "media": "windows"},  
,
```

Filepaths can be relative or absolute.

- Relative paths are resolved relative to the JSON form description file.
- For security reasons, only filesystem paths are accepted for absolute paths. (e.g., `"/RESOURCES"`, `"/DATA"`)

### See also

See the [CSS for 4D Forms](#) video presentation.

## Pictures

4D includes specific support for pictures used in your forms.

### Native Formats Supported

4D integrates native management of picture formats. This means that pictures will be displayed and stored in their original format, without any interpretation in 4D. The specific features of the different formats (shading, transparent areas, etc.) will be retained when they are copied and pasted, and will be displayed without alteration. This native support is valid for all pictures stored in 4D forms: [static pictures](#) pasted in Design mode, pictures pasted into [inputs objects](#) at runtime, etc.

The most common picture formats are supported of both platforms: .jpeg, .gif, .png, .tiff, .bmp, etc. On macOS, the .pdf format is also available for encoding and decoding.

The full list of supported formats varies according to the operating system and the custom codecs that are installed on the machines. To find out which codecs are available, you must use the `PICTURE CODEC LIST` command (see also the [picture data type](#) description).

### Unavailable picture format

A specific icon is displayed for pictures saved in a format that is not available on the machine. The extension of the missing format is shown at the bottom of the icon:



The icon is automatically used wherever the picture is meant to be displayed:

This icon indicates that the picture cannot be displayed or manipulated locally -- but it can be saved without alteration so that it can be displayed on other machines. This is the case, for example, for PDF pictures on Windows, or for PICT format pictures.

### High Resolution Pictures

4D supports high resolution pictures on both macOS and Windows platforms. High resolution pictures can be defined by either scale factor or dpi.

#### Scale factor

High resolution displays have a higher pixel density than traditional standard displays. For pictures to render correctly on high resolution displays, the number of pixels in the picture must be multiplied by the *scale factor* (*i.e.*, two times larger, three times larger, etc.).

When using high resolution pictures, you can specify the scale factor by adding "@nx" in the picture's name (where *n* designates the scale factor). In the table below, you can see that the scale factor is indicated in the names of the high resolution pictures, [\*circle@2x.png\*](#) and [\*circle@3x.png\*](#).

Display Type	Scale Factor	Example
Standard Resolution	1:1 pixel density.	<b>1x</b> <i>circle.png</i>
High Resolution	Pixel density increased by a factor of 2 or 3.	<b>2x</b> <i>circle@2x.png</i> <b>3x</b> <i>circle@3x.png</i>

High resolution pictures with the @nx convention can be used in the following objects:

- [Static pictures](#)
- [Buttons/radio/check boxes](#)
- [Picture buttons/Picture pop-ups](#)
- [Tab controls](#)
- [List box headers](#)
- [Menu icons](#)

4D automatically prioritizes pictures with the highest resolution. For example, when using two screens (one high resolution display, one standard display) and you move a form from one screen to another, 4D automatically renders the highest possible resolution of the picture. Even if a command or property specifies *circle.png*, *circle@3x.png* will be used (if it exists).

Note that resolution prioritization occurs only for displaying pictures onscreen, there is no automatic prioritization made when printing.

## DPI

While 4D automatically prioritizes the highest resolution, there are, however, some behavioral differences depending on screen and image dpi(\*), and picture format:

Operation	Behavior
Drop or Paste	<p>If the picture has:</p> <ul style="list-style-type: none"> <li>• <b>72dpi or 96dpi</b> - The picture is "<a href="#">Center</a>" formatted and the object containing the picture has the same number of pixels.</li> <li>• <b>Other dpi</b> - The picture is "<a href="#">Scaled to fit</a>" formatted and the object containing the picture is equal to (picture's number of pixels * screen dpi) / (picture's dpi)</li> <li>• <b>No dpi</b> - The picture is "<a href="#">Scaled to fit</a>" formatted.</li> </ul>
<a href="#">Automatic Size</a> (Form Editor context menu)	<p>If the picture's display format is:</p> <ul style="list-style-type: none"> <li>• <b>Scaled</b> - The object containing the picture is resized according to (picture's number of pixels * screen dpi) / (picture's dpi)</li> <li>• <b>Not scaled</b> - The object containing the picture has the same number of pixels as the picture.</li> </ul>

(\*) Typically, macOS = 72dpi, Windows = 96dpi

## Dark mode pictures (macOS only)

You can define specific pictures and icons to be used instead of standard pictures when [forms use the dark scheme](#).

A dark mode picture is defined in the following way:

- dark mode picture has the same name as the standard (light scheme) version

with the suffix "`_dark`"

- dark mode picture is stored next to the standard version.

At runtime, 4D will automatically load the light or dark image according to the [current form color scheme](#).

## Mouse Coordinates in a Picture

4D lets you retrieve the local coordinates of the mouse in an [input object](#) associated with a [picture expression](#), in case of a click or a hovering, even if a scroll or zoom has been applied to the picture. This mechanism, similar to that of a picture map, can be used, for example, to handle scrollable button bars or the interface of cartography software.

The coordinates are returned in the *MouseX* and *MouseY* [System Variables](#). The coordinates are expressed in pixels with respect to the top left corner of the picture (0,0). If the mouse is outside of the picture coordinates system, -1 is returned in *MouseX* and *MouseY*.

You can get the value of these variables as part of the [On Clicked](#), [On Double Clicked](#), [On Mouse up](#), [On Mouse Enter](#), or [On Mouse Move](#) form events.

# Form Editor

4D provides a full-featured Form editor that allows you to modify your form until you achieve the effect that you want. With the Form editor, you can create and delete form objects, manipulate them directly, and set form and object properties.

## Interface

The Form editor interface displays each JSON form in its own window, which has both an object and tool bar. You can have several forms open at the same time.

### Display options

You can show or hide several interface elements on the current page of the form:

- **Inherited Form:** Inherited form objects (if there is an [inherited form](#)).
- **Page 0:** Objects from [page 0](#). This option allows you to distinguish between the objects on the form's current page and those on page 0.
- **Paper:** Borders of the printing page, which are shown as gray lines. This element can only be displayed by default in ["for printing" type](#) forms.
- **Rulers:** Rulers of the Form editor's window.
- **Markers:** Output control lines and associated markers that show the limits of the form's different areas. This element can only be displayed by default in [list forms](#).
- **Marker Labels:** Marker labels, available only when the output control lines are displayed. This element can only be displayed by default in [list forms](#).
- **Limits:** Form's limits. When this option is selected, the form is displayed in the Form editor as it appears in Application mode. This way you can adjust your form without having to switch to the Application mode in order to see the result.

The [Size Based on](#), [Hor margin](#) and [Vert margin](#) settings of the form properties affect the form's limits. When using these settings, the limits are based on the objects in the form. When you modify the size of an object that is located next to the form's border, it is modified to reflect that change.

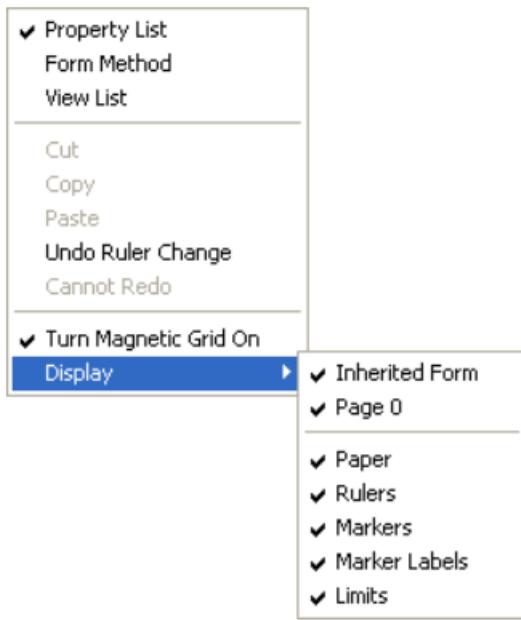
### Default display

When a form is opened in the editor, interface elements are displayed or hidden by default, depending on:

- the **New form default display** options set in the Preferences - unchecked options cannot be displayed by default.
- the current [form type](#):
  - Markers and marker labels are always displayed by default on list forms
  - Paper is displayed by default on "for printing" forms.

### Display/Hide elements

You can display or hide elements at any moment in the Form editor's current window by selecting **Display** from the **Form** menu or the Form editor's context menu:



## Rulers

The rulers on the side and bottom help you position objects in the form. They can be [displayed or hidden](#).

Select **Ruler definition...** from the **Form** menu to change measurement units so that the form displays inches, centimeters, or pixels.

## Toolbar

The toolbar of the Form editor offers a set of tools to manipulate and modify the form. Each window has its own toolbar.



The toolbar contains the following elements:

Icon	Name	Description
▶	Execute the form	Used to test the execution of the form. When you click on this button, 4D opens a new window and displays the form in its context (list of records for a list form and current record page for a detail form). The form is executed in the main process.
		Allows selecting, moving and resizing form objects.
	<a href="#">Selection tool</a>	<b>Note:</b> When an object of the Text or Group Box type is selected, pressing the <b>Enter</b> key lets you switch to editing mode.
⇄	<a href="#">Entry order</a>	Switches to "Entry order" mode, where it is possible to view and change the current entry order of the form. Note that shields allow viewing the current entry order, while still working in the form.
	<a href="#">Moving</a>	Switches to "Move" mode, where it is possible to reach any part of the form quickly by using drag and drop in the window. The cursor takes the shape of a hand. This navigation mode is particularly useful when zooming in the form.
	<a href="#">Zoom</a>	Allows modifying the form display percentage (100% by default). You can switch to "Zoom" mode by clicking on the magnifying glass or by clicking directly on the desired bar. This feature is detailed in previous section.
----	<a href="#">Alignment</a>	This button is linked to a menu that allows aligning objects in the form. It is enabled (or not) depending on the objects selected.
		Available only with CSS Preview None
----	<a href="#">Distribution</a>	This button is linked to a menu that allows distributing objects in the form. It is enabled (or not) depending on the objects selected.
		Available only with CSS Preview None
----	<a href="#">Level</a>	This button is linked to a menu that allows changing the level of objects in the form. It is enabled (or not) depending on the objects selected.
----	<a href="#">Group/Ungroup</a>	This button is linked to a menu that allows grouping and ungrouping selections of objects in the form. It is enabled (or not) depending on the objects selected.
----	<a href="#">Display and page management</a>	This area allows passing from one form page to another and adding pages. To navigate among form pages, click the arrow buttons, or click the central area and choose the page to display from the menu that appears. If you click the right arrow button while the last form page is displayed, 4D allows you to add a page.
	<a href="#">CSS Preview</a>	This button is used to select the CSS Mode to use.
	<a href="#">Managing views</a>	This button displays or hides the views palette. This function is detailed in Using object views .
	<a href="#">Displaying shields</a>	Each click on this button causes the successive display of each type of form shield. The button is also linked to a menu that allows directly selecting the type of shield to display.
	<a href="#">Preconfigured object library</a>	This button displays the preconfigured object library that provides numerous objects with certain properties that have been predefined.
	<a href="#">List Box Builder</a>	This button creates new entity selection list boxes.

## Object bar

The object bar contains all the active and inactive objects that can be used in 4D forms. Some objects are grouped together by themes. Each theme includes several alternatives that you can choose between. When the object bar has the focus, you can select the buttons using the keys of the keyboard. The following table describes the object groups available and their associated shortcut key.

Button	Group	Key
 <a href="#">Text / Group Box</a>		T
 <a href="#">Input</a>		F
 <a href="#">Hierarchical List / List Box</a>		L
 <a href="#">Combo Box / Drop-down List / Picture Pop-up Menu</a>		P
 <a href="#">Button / Picture Button / Button Grid</a>		B
 <a href="#">Radio Button</a>		R
 <a href="#">Check Box</a>		C
 <a href="#">Progress Indicator / Ruler / Stepper / Spinner</a>		I
 <a href="#">Rectangle / Line / Oval</a>		S
 <a href="#">Splitter / Tab Control</a>		D
 <a href="#">Plug-in Area / Subform / Web Area / 4D Write Pro / 4D View Pro</a>		X

To draw an object type, select the corresponding button and then trace the object in the form. After creating an object, you can modify its type using the Property List. Hold down the **Shift** key as you draw to constrain the object to a regular shape. Lines are constrained to horizontal, 45°, or vertical, rectangles are constrained to squares, and ovals are constrained to circles.

The current variant of the theme is the object that will be inserted in the form. When you click the right side of a button, you access the variant menu:



You can click twice on the button so that it remains selected even after you have traced an object in the form (continual selection). This function makes creating several successive objects of the same type easier. To cancel a continual selection, click on another object or tool.

## Property List

Both forms and form objects have properties that control access to the form, the appearance of the form, and the behavior of the form when it is used. Form properties include, for example, the form's name, its menu bar, and its size. Object Properties include, for example, an object's name, its dimensions, its background color, and its font.

You can display and modify form and object properties using the Property List. It displays either form or objects properties depending on what you select in the editor window.

To display/hide the Property List, choose **Property List** from the **Form** menu or from the context menu of the Form editor. You can also display it by double-clicking in an empty area of the form.

## Shortcuts

You can use the following shortcuts in the Property List:

- **Arrow keys** ↑ ↓: Used to go from one cell to another.
- **Arrow keys** ← →: Used to expand/collapse themes or enter edit mode.
- **PgUp** and **PgDn**: Used to scroll the Property List contents.
- **Home** and **End**: Used to scroll the Property List so that the first or last cell is displayed.
- **Ctrl+click** (Windows) or **Command+click** (macOS) on an event: Used to select/deselect every event in the list, according to the initial state of the event on which you clicked.
- **Ctrl+click** (Windows) or **Command+click** (macOS) on a theme label: Used to Collapse/Expand every theme in the list.
- **Ctrl+click** (Windows) or **Command+click** (macOS) on a property value displayed in **bold**: Resets the property to its default.

## Manipulating Form Objects

### Adding objects

You can add objects to forms in several ways:

- By drawing the object directly in the form after selecting its type in the object bar (see [Using the object bar](#))
- By dragging and dropping the object from the object bar
- By drag-and-drop or copy-paste operations on an object selected from the preconfigured [object library](#),
- By dragging and dropping an object from another form,
- By dragging and dropping an object from the Explorer (fields) or from other editors in the Design environment (lists, pictures, etc.)

Once the object is placed in the form, you can modify its characteristics using the Form editor.

You can work with two types of objects in your forms:

- **Static objects** (lines, frames, background pictures, etc.): These objects are generally used for setting the appearance of the form and its labels as well as for the graphic interface. They are available in the object bar of the Form editor. You can also set their graphic attributes (size, color, font, etc.) and their resizing properties using the Property List. Static objects do not have associated variables like active objects. However, you can insert dynamic objects into static objects.
- **Active objects**: These objects perform tasks or functions in the interface and can take many forms: fields, buttons, scrollable lists, etc. Each active object is associated with either a field or a variable.

### Selecting objects

Before you can perform any operation on an object (such as changing a line width or

font), you need to select the object that you want to modify.

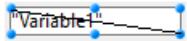
To select an object using the toolbar:

1. Click the Arrow tool in the toolbar.



When you move the pointer into the form area, it becomes a standard arrow-shaped pointer

2. Click the object you want to select. Resizing handles identify the selected object.



To select an object using the Property List:

1. Choose the object's name from the Object List drop-down list located at the top of the Property List. Using these two methods, you can select an object that is hidden by other objects or located outside the visible area of the current window. To deselect an object, click outside the object's boundary or **Shift+click** the object.

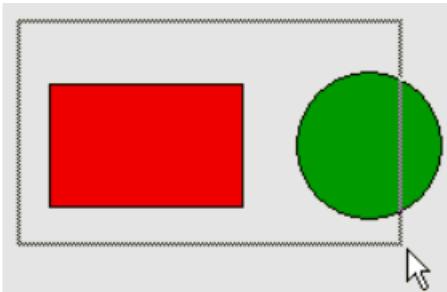
It is also possible to select objects by double-clicking them in the result window of ""Find in design" operation.

## Selecting multiple objects

You may want to perform the same operation on more than one form object — for example, to move the objects, align them, or change their appearance. 4D lets you select several objects at the same time. There are several ways to select multiple objects:

- Choose **Select All** from the Edit menu to select all the objects.
- Right-click on the object and choose the **Select Similar Objects** command in the context menu.
- Hold down the **Shift** key and click the objects you want to select.
- Start at a location outside the group of objects you want to select and drag a marquee (sometimes called a selection rectangle) around the objects. When you release the mouse button, if any part of an object lies within or touches the boundaries of the selection rectangle, that object is selected.
- Hold down the **Alt** key (Windows) or the **Option** key (macOS) and draw a marquee. Any object that is completely enclosed by the marquee is selected.

The figure below shows a marquee being drawn to select two objects:



To deselect an object that is part of a set of selected objects, hold down the **Shift** key

and click the object. The other objects remain selected. To deselect all the selected objects, click outside the boundaries of all the objects.

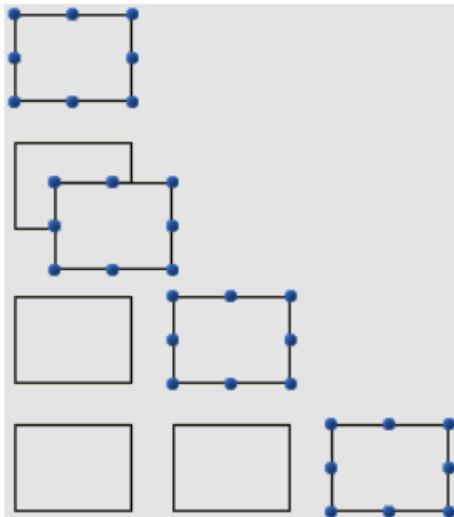
## Duplicating objects

You can duplicate any object in the form, including active objects. Copies of active objects retain all the properties of the original, including name, type, standard action, display format, and object method.

You can duplicate an object directly using the Duplicate tool in the Tools palette or use the Duplicate Many dialog box to duplicate an object more than once. Also, using this dialog box, you can set the distance between two copies.

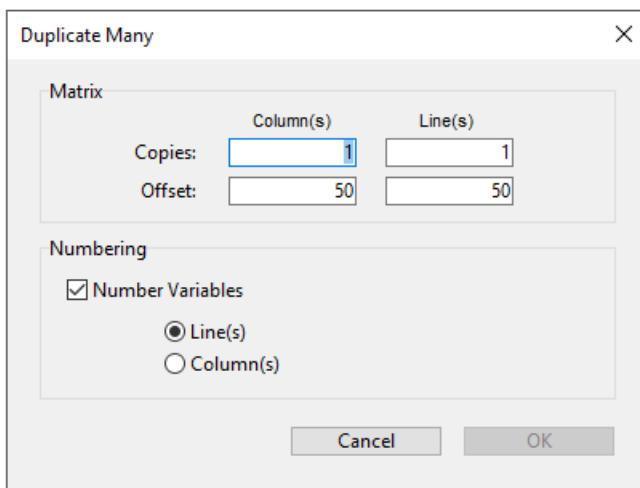
To duplicate one or more objects:

1. Select the object or objects that you want to duplicate.
2. Choose **Duplicate** from the **Edit** menu. 4D creates a copy of each selected object and places the copy in front and slightly to the side of the original.
3. Move the copy (or copies) to the desired location. If you choose the Duplicate menu item again, 4D creates another copy of each object and moves it the exact same distance and direction from the first copy. If you need to distribute copies of the object along a line, you should use the following procedure. Duplicate the original object, move the copy to another location in the form, and then duplicate the copy. The second copy is automatically placed in the same relation to the first copy as the first copy was in relation to the original object. Subsequent copies are also placed in the same relation to their originals. The figure below shows how this relative placement of copies works:



## Duplicate Many

The "Duplicate Many" dialog box appears when you select one or more object(s) and choose the **Duplicate Many...** command from the **Object** menu.



- In the upper area, enter the number of columns and lines (rows) of objects you want to get. For example, if you want three columns and two lines of objects, enter 3 in the Column(s) area and 2 in the Line(s) area. If you want three horizontal new copies of an object, enter 4 in the Column(s) area and leave the default value, 1, in the Line(s) area.
- For lines and columns, define the offset that you wish to leave between each copy. The value must be expressed in points. It will be applied to each copy, or copies, in relation to the original object. For example, if you want to leave a vertical offset of 20 points between each object and the height of the source object is 50 points, enter 70 in the column's "Offset" area.
- If you wish to create a matrix of variables, select the **Number Variables** option and select the direction in which the variables are to be numbered, either by line(s) or by column(s). This option is active only when the selected object is a variable. For more information on this option, refer to **Duplicating on a matrix** in the *Design Reference*.

## Moving objects

You can move any graphic or active object in the form including fields and objects created with a template. When moving an object, you have the following options:

- Move the object by dragging it,
- Move the object one pixel at a time using the arrow keys,
- Move the object by steps using the arrow keys (20-pixel steps by default),

As you begin dragging the selected object, its handles disappear. 4D displays markers that show the location of the object's boundaries in the rulers so that you can place the object exactly where you want it. Be careful not to drag a handle. Dragging a handle resizes the object. You can press the **Shift** key to carry out the move with a constraint.

When the **Magnetic Grid** is on, objects are moved in stages indicating noticeable locations.

To move an object one pixel at a time:

- Select the object or objects and use the arrow keys on the keyboard to move the object. Each time you press an arrow key, the object moves one pixel in the direction of the arrow.

To move an object by steps:

- Select the object or objects you want to move and hold down the **Shift** key and use the arrow keys to move the object by steps. By default, steps are 20 pixels at a time. You can change this value on the Forms Page of the Preferences.

## Grouping objects

4D lets you group objects so that you can select, move, and modify the group as a single object. Objects that are grouped retain their position in relation to each other. You would typically group a field and its label, an invisible button and its icon, and so forth.

When you resize a group, all the objects in the group are resized proportionally (except text areas, which are resized in steps according to their font sizes).

You can ungroup a group of objects to treat them as individual objects again.

An active object that has been grouped must be ungrouped before you can access its properties or method. However, it is possible to select an object belonging to a group without degrouping the set: to do this, **Ctrl+click** (Windows) or **Command+click** (macOS) on the object (the group must be selected beforehand).

Grouping only affects objects in the Form editor. When the form is executed, all grouped objects act as if they were ungrouped.

It is not possible to group objects belonging to different views and only those objects belonging to the current view can be grouped (see [Views](#) ).

To group objects:

1. Select the objects that you want to group.
2. Choose **Group** from the Object menu. OR Click the Group button in the toolbar of the Form editor:



4D marks the boundary of the newly grouped objects with handles. No handles mark the boundary of any of the individual objects within the group. Now, when you modify the grouped object, you change all the objects that make up the group.

To ungroup an object:

1. Select the grouped object that you want to ungroup.
2. Choose **Ungroup** from the Object menu.

OR

Click the **Ungroup** button (variant of the **Group** button) in the toolbar of the Form editor.

If **Ungroup** is dimmed, this means that the selected object is already separated into its simplest form.

4D marks the boundaries of the individual objects with handles.

## Aligning objects

You can align objects with each other or using an invisible grid on the form.

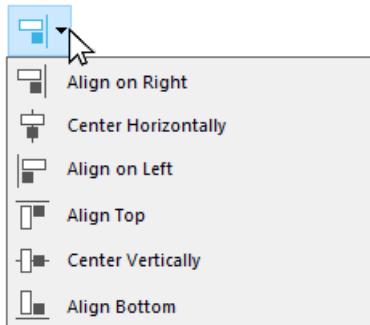
- When you align one object to another, you can align it to the top, bottom, side, or

horizontal or vertical center of the other object. You can directly align a selection of objects using the alignment tools or apply more advanced alignment settings using the Alignment Assistant. The latter option allows you, for example, to set the object that will be used as the position reference and to preview the alignment in the form before applying it.

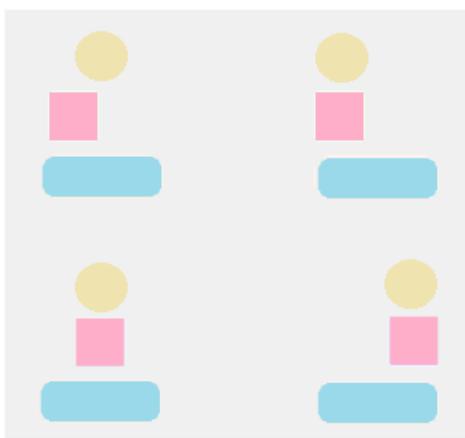
- When you use the invisible grid, each object can be aligned manually with others based on “noticeable” positions which are depicted with dotted lines that appear when the object being moved approaches other objects.

## Using the instantaneous alignment tools

The alignment tools in the toolbar and in the Align submenu of the Object menu allow you to quickly align selected objects.

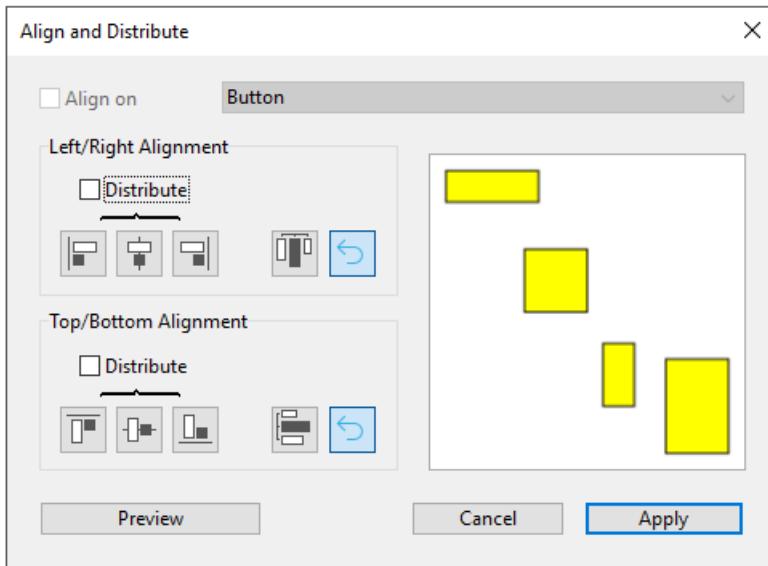


When 4D aligns objects, it leaves one selected object in place and aligns the remaining objects to that one. This object is the “anchor.” It uses the object that is the furthest in the alignment’s direction as the anchor and aligns the other objects to that object. For instance, if you want to perform a right alignment on a set of objects, the rightmost object will be used as the anchor. The figure below shows objects with no alignment, “aligned left”, “aligned horizontally by centers”, and “aligned right”:



## Using the alignment assistant

The Alignment Assistant allows you to perform any type of alignment and/or distribution of objects.



To display this dialog box, select the objects you want to align then choose the **Alignment** command from the **Align** submenu in the **Object** menu or from the context menu of the editor.

- In the “Left/Right Alignment” and/or “Top/Bottom Alignment” areas, click the icon that corresponds to the alignment you want to perform.

The example area displays the results of your selection.

- To perform an alignment that uses the standard anchor scheme, click **Preview** or **Apply**. In this case 4D uses the object that is the furthest in the alignment’s direction as the anchor and aligns the other objects to that object. For instance, if you want to perform a right alignment on a set of objects, the rightmost object will be used as the anchor. OR:

To align objects to a specific object, select the **Align on** option and select the object to which you want the other objects to be aligned from the object list. In this case, the position of the reference object will not be altered.

You can preview the results of the alignment by clicking the **Preview** button. The objects are then aligned in the Form editor but since the dialog box does not go away, you can still cancel or apply the alignment.

This dialog box allows you to align and distribute objects in one operation.

For more information on how to distribute objects, refer to [Distributing objects](#).

## Using the Magnetic Grid

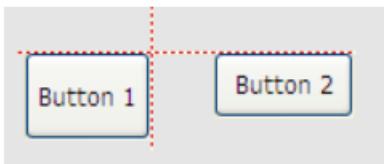
The Form editor provides a virtual magnetic grid that can help you place and align objects in a form. Magnetic alignment of objects is based on their position in relation to each other. The magnetic grid can only be used when at least two objects are present in the form.

This works as follows: When you move an object in the form, 4D indicates possible locations for this object based on noticeable alignments with other form objects. A noticeable alignment is established each time that:

- Horizontally, the edges or centers of two objects coincide,
- Vertically, the edges of two objects coincide.

When this happens, 4D places the object at the location and displays a red line

indicating the noticeable alignment taken into account:



Concerning the distribution of objects, 4D proposes a distance based on interface standards. Like with magnetic alignment, red lines indicate the noticeable differences once they are reached.



This operation applies to all types of form objects. The Magnetic Grid can be enabled or disabled at any time using the **Magnetic Grid** command in the **Form** menu or in the editor context menu. It is also possible to set the activation of this feature by default on the **Preferences > Forms** page (**Activate auto alignment by default** option). You can manually activate or deactivate the magnetic grid when an object is selected by pressing the **Ctrl** (Windows) or **Control** (macOS) key .

The Magnetic Grid also influences the manual resizing of objects.

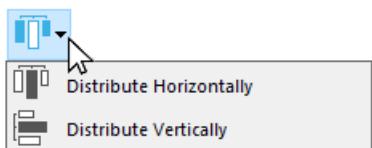
## Distributing objects

You can distribute objects so that they are set out with an equal amount of space between them. To do this, you can distribute objects using either the Distribute tools in the Tools palette or the Alignment Assistant. The latter allows you to align and distribute objects in one operation.

When the [Magnetic Grid](#) is on, a visual guide is also provided for distribution when an object is moved manually.

To distribute objects with equal spacing:

1. Select three or more objects and click the desired Distribute tool.
2. In the toolbar, click on the distribution tool that corresponds to the distribution you want to apply.



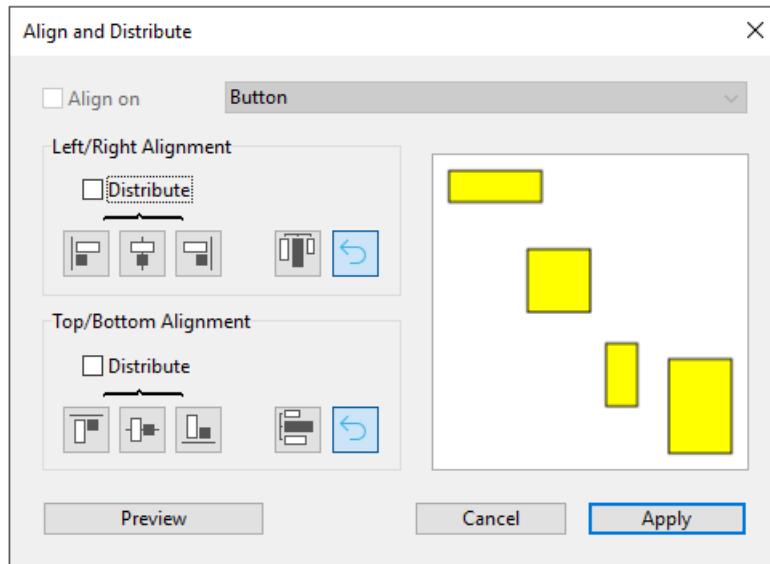
OR

Select a distribution menu command from the **Align** submenu in the **Object** menu or from the context menu of the editor.

4D distributes the objects accordingly. Objects are distributed using the distance to their centers and the largest distance between two consecutive objects is used as a reference.

To distribute objects using the Align and Distribute dialog box:

1. Select the objects you want to distribute.
2. Choose the **Alignment** command from the **Align** submenu in the **Object** menu or from the context menu of the editor. The following dialog box appears:



3. In the Left/Right Alignment and/or Top/Bottom Alignment areas, click the standard distribution icon:
- (Standard horizontal distribution icon)
- The example area displays the results of your selection.

4. To perform a distribution that uses the standard scheme, click **Preview** or **Apply**.

In this case 4D will perform a standard distribution, so that the objects are set out with an equal amount of space between them.

OR:

To execute a specific distribution, select the **Distribute** option (for example if you want to distribute the objects based on the distance to their right side). This option acts like a switch. If the Distribute check box is selected, the icons located below it perform a different function:

- Horizontally, the icons correspond to the following distributions: evenly with respect to left sides, centers (hor.) and right sides of the selected objects.
- Vertically, the icons correspond to the following distributions: evenly with respect to top edges, centers (vert.) and bottom edges of the selected objects.

You can preview the actual result of your settings by clicking on the **Preview** button: the operation is carried out in the Form editor but the dialog box stays in the foreground. You can then **Cancel** or **Apply** the modifications.

This dialog box lets you combine object alignment and distribution. For more information about alignment, refer to [Aligning objects](#).

## Layering objects

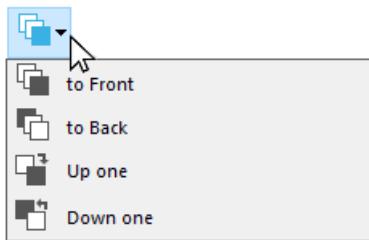
You will sometimes have to rearrange objects that are obstructing your view of other objects in the form. For example, you may have a graphic that you want to appear behind the fields in a form. 4D provides four menu items, **Move to Back**, **Move to Front**, **Up One Level** and **Down One Level** that let you “layer” objects on the form.

These layers also determine the default entry order (see [Modifying data entry order](#)). The figure below shows objects in front of and behind other objects:



To move an object to another level, select it and choose:

- One of the **Move to Back**, **Move to Front**, **Up One Level** and **Down One Level** commands of the Object menu,
- One of the commands in the **Level>** submenu in the context menu of the editor,
- One of the commands associated with the level management button of the toolbar.



When several objects are superimposed, the **Ctrl+Shift+click** / **Command+Shift+click** shortcut can be used to select each object successively by going down a layer with each click.

When ordering different levels, 4D always goes from the background to the foreground. As a result, the previous level moves the selection of objects one level towards the background. The next level moves the selection one level towards the foreground of the form.

## Data entry order

The data entry order is the order in which fields, subforms, and other active objects are selected as you hit the **Tab** or the **Carriage return** key in an input form. It is possible to move through the form in the opposite direction (reverse data entry order) by pressing the **Shift+Tab** or **Shift+Carriage return** keys.

You can change the entry order at runtime using the `FORM SET ENTRY ORDER` and `FORM GET ENTRY ORDER` commands.

Every object that supports the `focusable` property is included in the data entry order by default.

Setting the entry order for a JSON form is done with the `entryOrder` property.

If you don't specify a custom entry order, by default 4D uses the layering of the objects to determine the entry order in the direction "background towards foreground." The standard entry order thus corresponds to the order in which the objects were created in the form.

In some forms, a custom data entry order is needed. Below, for example, additional fields related to the address have been added after the creation of the form. The resulting standard entry order thus becomes illogical and forces the user to enter the information in an awkward manner:

Last Name : [People]Last Name  
First Name : [People]First Name  
Full address : [People]Address  
Telephone : [People]Phone  
Company : [People]Company  
Hire date : [People]hire

In cases such as this, a custom data entry order allows you to enter the information in a more logical order:

Last Name : [People]Last Name  
First Name : [People]First Name  
Full address : [People]Address  
Telephone : [People]Phone  
Company : [People]Company  
Hire date : [People]hire

## Viewing and changing the data entry order

You can view the current entry order either using the “Entry order” shields, or by using the “Entry order” mode. However, you can only modify the entry order using the “Entry order” mode.

This paragraph describes viewing and modifying the entry order using the “Entry order” mode. For more information about viewing the entry order using shields, refer to [Using shields](#).

To view or change the entry order:

1. Choose **Entry Order** from the **Form** menu or click on the Entry Order button in the toolbar of the window:



The pointer turns into an entry order pointer and 4D draws a line in the form showing the order in which it selects objects during data entry. Viewing and changing the data entry order are the only actions you can perform until you click any tool in the Tools palette.

2. To change the data entry order, position the pointer on an object in the form and, while holding down the mouse button, drag the pointer to the object you want next in the data entry order.

4D will adjust the entry order accordingly.

3. Repeat step 2 as many times as necessary to set the data entry order you want.
4. When you are satisfied with the data entry order, click any unselected tool in the toolbar or choose **Entry Order** from the **Form** menu. 4D returns to normal operation of the Form editor.

Only the entry order of the current page of the form is displayed. If the form contains enterable objects on page 0 or coming from an inherited form, the default entry order is as follows: Objects from page 0 of the inherited form > Objects from page 1 of the inherited form > Objects from page 0 of the open form > Objects from the current page of the open form.

### Using a data entry group

While you are changing the data entry order, you can select a group of objects in a form so that the standard data entry order applies to the objects within the group. This allows you to easily set the data entry order on forms in which fields are separated into groups or columns.

To create a data entry group:

1. Choose **Entry Order** from the *Form* menu or click the button in the toolbar.
2. Draw a marquee around the objects you want to group for data entry.

When you release the mouse button, the objects enclosed or touched by the rectangle follow the standard data entry order. The data entry order for the remaining objects adjusts as necessary.

### Excluding an object from the entry order

By default, all objects that support the focusable property are included in the entry order. To exclude an object from the entry order:

1. Select the Entry order mode, then
2. **shift-click** on the object
3. **right-click** on the object and select **Remove from entry order** option from the context menu

## CSS Preview

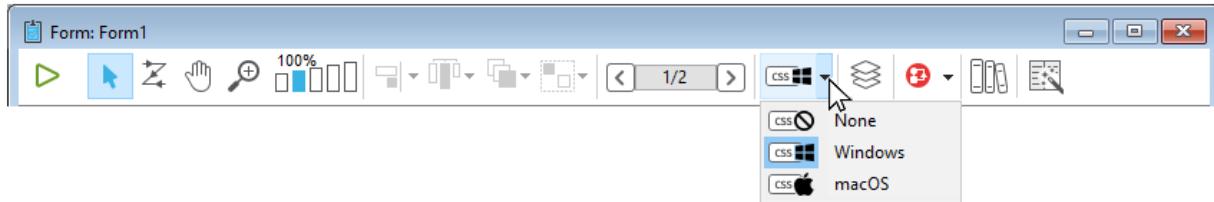
The Form editor allows you to view your forms with or without applied CSS values.

When [style sheets](#) have been defined, forms (including inherited forms and subforms)

are opened in the CSS Preview mode for your operating system by default.

## Selecting CSS Preview Mode

The Form editor toolbar provides a CSS button for viewing styled objects:



Select one of the following preview modes from the menu:

Toolbar Icon	CSS Preview Mode	Description
	None	No CSS values are applied in the form and no CSS values or icons displayed in the Property List.
	Windows	CSS values for Windows platform are applied in the form. CSS values and icons displayed in the Property List.
	macOS	CSS values for macOS platform are applied in the form. CSS values and icons displayed in the Property List.

If a font size too large for an object is defined in a style sheet or JSON, the object will automatically be rendered to accommodate the font, however the size of the object will not be changed.

The CSS preview mode reflects the priority order applied to style sheets vs JSON attributes as defined in the [JSON vs Style Sheet](#) section.

Once a CSS preview mode is selected, objects are automatically displayed with the styles defined in a style sheet (if any).

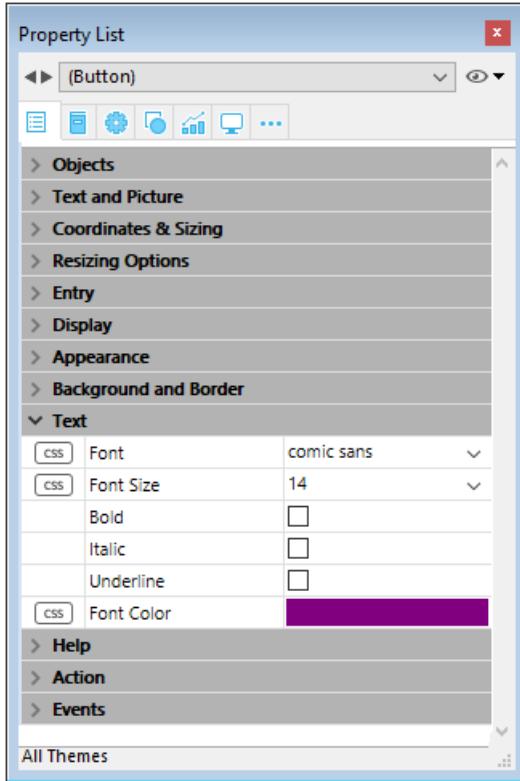
When copying or duplicating objects, only the CSS references (if any) and the JSON values are copied.

## CSS support in the Property List

In CSS Preview mode, if the value of an attribute has been defined in a style sheet, the attribute's name will appear with a CSS icon displayed next to it in the Property List. For example, the attribute values defined in this style sheet:

```
.myButton {  
font-family: comic sans;  
font-size: 14;  
stroke: #800080;  
}
```

are displayed with a CSS icon in the Property List:



An attribute value defined in a style sheet can be overridden in the JSON form description (except if the CSS includes the `!important` declaration, see below). In this case, the Property List displays the JSON form value in **bold**. You can reset the value to its style sheet definition with the **Ctrl + click** (Windows) or **Command + click** (macOs) shortcuts.

If an attribute has been defined with the `!important` declaration for a group, an object within a group, or any object within a selection of multiple objects, that attribute value is locked and cannot be changed in the Property List.

## Property List CSS Icons

Icon	Description
	Indicates that an attribute value has been defined in a style sheet
	Indicates that an attribute value has been defined in a style sheet with the <code>!important</code> declaration
	Displayed when an attribute value defined in a style sheet for at least one item in a group or a selection of multiple objects is different from the other objects

## List Box Builder

You can create new entity selection list boxes quickly with the **List box builder**. The new list box can be used immediately or it can be edited via the Form Editor.

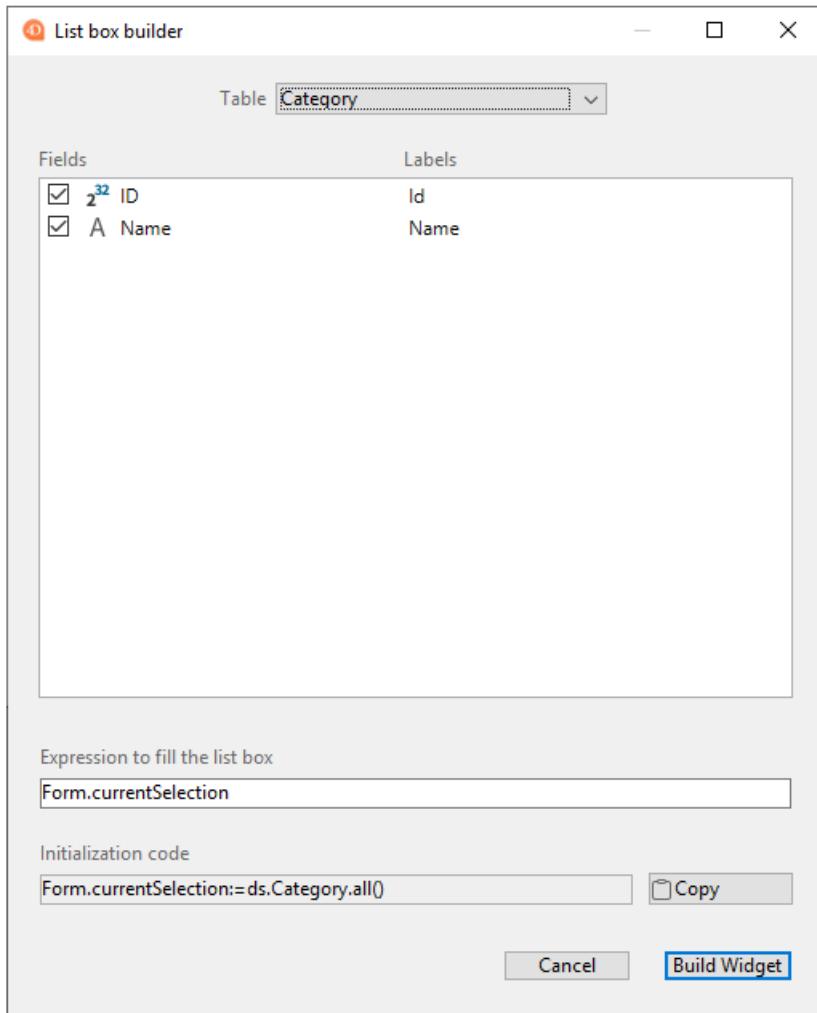
The List box builder lets you create and fill entity selection list boxes in a few simple operations.

### Using the List Box Builder

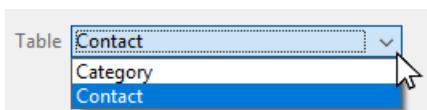
1. In the Form Editor toolbar, click on the List box builder icon:



The List box builder is displayed:



2. Select a table from the **Table** dropdown list:



3. Select the fields for the list box in the **Fields** area:

Fields		Labels
<input type="checkbox"/>	<sup>32</sup> A CategoryID	Category id
<input type="checkbox"/>	A Email	Email
<input checked="" type="checkbox"/>	A FirstName	First Name
<input type="checkbox"/>	A HomeAddress	Home Address
<input type="checkbox"/>	A HomeCity	Home City
<input type="checkbox"/>	A HomeCountry	Home Country
<input type="checkbox"/>	A HomePhone	Home Phone
<input type="checkbox"/>	A HomeState	Home State
<input type="checkbox"/>	A HomeZip	Home zip
<input type="checkbox"/>	<sup>32</sup> A ID	Id
<input type="checkbox"/>	A JobTitle	Job Title
<input checked="" type="checkbox"/>	A LastName	Last Name
<input type="checkbox"/>	A MiddleName	Middle Name
<input type="checkbox"/>	A MobilePhone	Mobile Phone
<input type="checkbox"/>	A Notes	Notes
<input checked="" type="checkbox"/>	A Organization	Organization

By default, all fields are selected. You can select or deselect fields individually or use **Ctrl+click** (Windows) or **Cmd+click** (macOS) to select or deselect them all at once.

You can change the order of the fields by dragging them and dropping them.

4. The expression to fill the list box's rows from the entity selection is prefilled:

Expression to fill the list box

This expression can be changed if necessary.

5. Clicking on the **Copy** button will copy the expression for loading all records into memory:

Initialization code

```
Form.currentSelection:=ds.Contact.all()
```

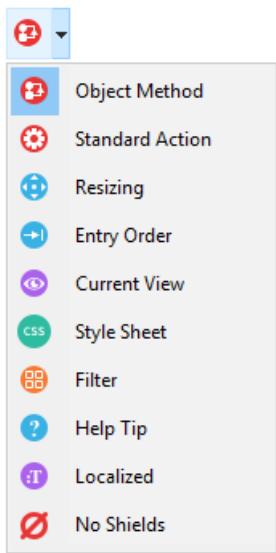
6. Click the the **Build widget** button to create the list box.

### Build widget

The final list box:

# Shields

The 4D Form Editor uses shields to make viewing object properties easier. You can find them on the form toolbar:



This function works as follows: Each shield is associated with a property (for example, **Views**, which means the object “is in the current view”). When you activate a shield, 4D displays a small icon (shield) in the upper left of each object of the form where the property is applied.



## Using shields

To activate a shield, click the *Shield* icon from the toolbar until the desired shield is selected. You can also click on the right side of the button and select the type of shield to display directly in the associated menu:

If you don't want to display shields, select **No Shields** in the selection menu.

You can set which shields to display by default on the Forms Page of the application Preferences.

## Shield descriptions

Here is a description of each type of shield:

Icon	Name	Is displayed ...
	Object Method	For objects with an associated object method
	Standard Action	For objects with an associated standard action
	Resizing	For objects with at least one resizing property, indicates the combination of current properties
	Entry Order	For enterable objects, indicates the number of entry order
	Current View	For all objects in the current view
	Style Sheet	For objects with one or more attribute values overridden by a style sheet.
	Filter	For enterable objects with an associated entry filter
	Help Tip	For objects with an associated tip
	Localized	For objects whose label comes from a reference (label beginning with ":"). The reference can be of the resource (STR#) or XLIFF type
	No Shields	No shields appear

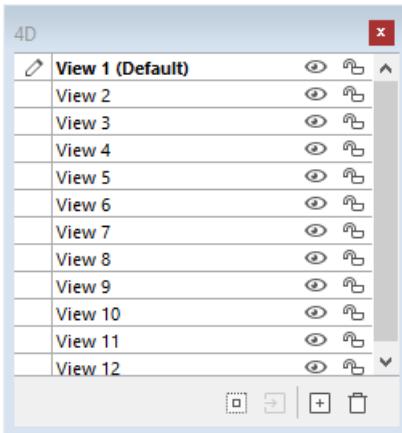
## Views

The 4D Form Editor enables you to build complex forms by distributing form objects among separate views that can then be hidden or shown as needed.

For example, you can distribute objects according to type (fields, variables, static objects, etc.). Any type of form object, including subforms and plug-in areas, can be included in views.

There is no limit on the number of views per form. You can create as many different views as you need. Additionally, each view can be displayed, hidden, and/or locked.

View management is handled via the View palette.



## Accessing the View palette

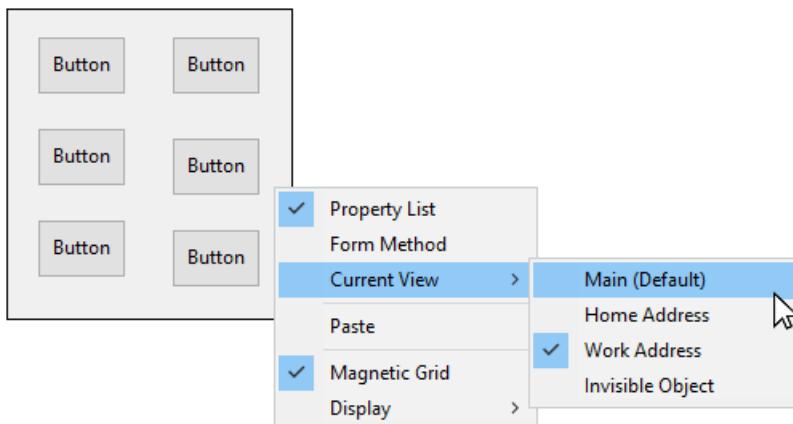
There are three ways to access the View palette:

- **Toolbar:** Click on the Views icon in the Form Editor toolbar. (This icon appears gray when at least one object belongs to a view other than the default view.)

## **Default view only With additional views**

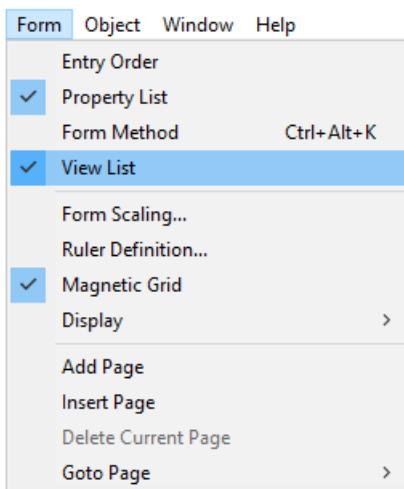


- **Context menu** (form or object): Right-click anywhere in the Form Editor or an object, and select **Current View**



The current view is indicated with a check mark (e.g., "Work Address" in the image above)

- **Form menu:** Click on the **Form** menu and select **View List**



## Before you begin

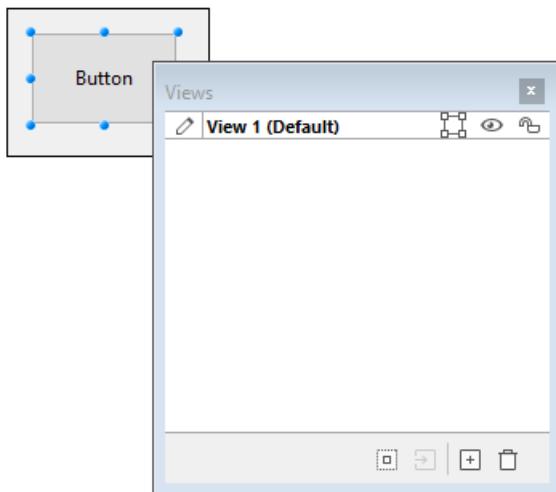
Here are a few important things to know before you start working with views:

- **Context of use:** Views are a purely graphic tool which can only be used in the Form Editor; you cannot access views programmatically or in the Application environment.
- **Views and pages:** Objects of the same view can belong to different form pages; only objects of the current page (and of page 0 if it is visible) can be displayed, regardless of the view configuration.
- **Views and levels:** Views are independent of object levels; there is no display hierarchy among different views.
- **Views and groups:** Only objects belonging to the current view can be grouped.
- **Current and Default** views: The Default view is the first view of a form and cannot be deleted; the Current view is the view that is being edited and the name is displayed in bold text.

## Managing views

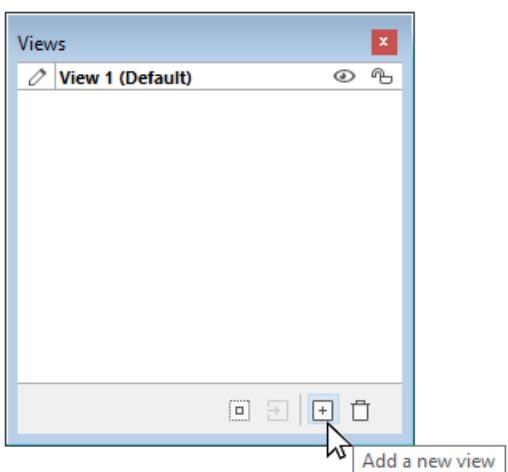
## Creating views

Any object created in a form is placed in the first view ("View 1") of the form. The first view is **always** the default view, indicated by (Default) after the name. The view's name can be changed (see [Renaming views](#)), however it remains the default view.

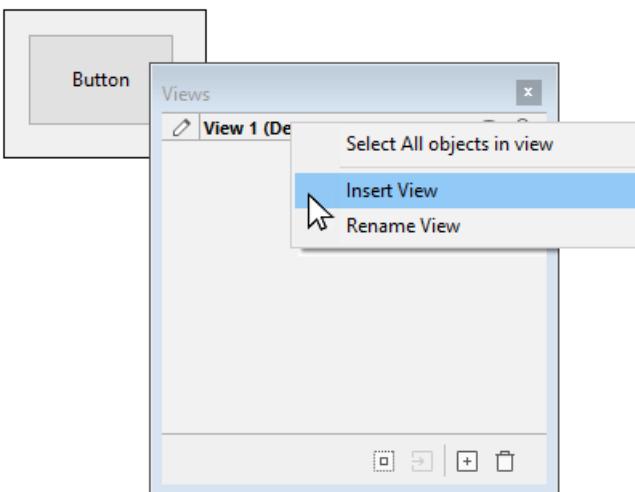


There are two ways to add additional views:

- Click on the **Add a new view** button at the bottom of the View palette:



- Right-click on an existing view and select **Insert view**:



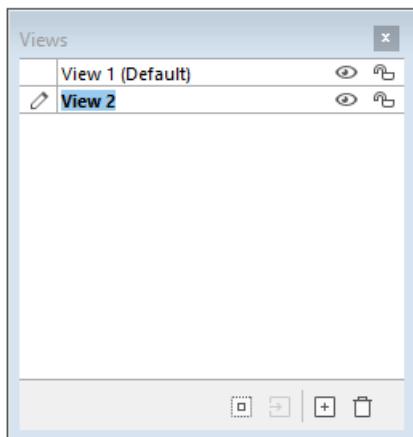
There is no limitation on the number of views.

## Renaming views

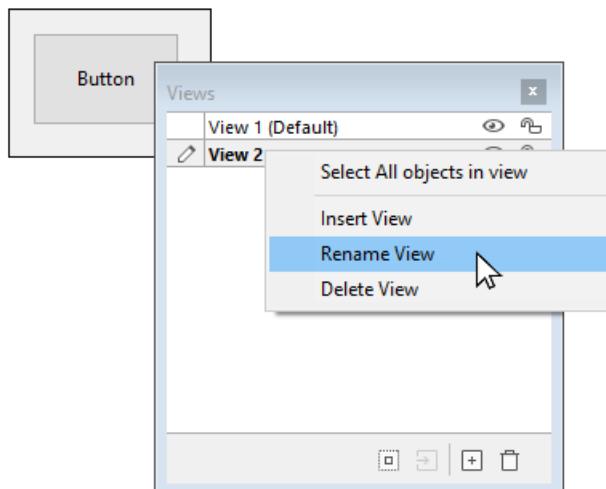
By default views are named as "View" + the view number, however you can change these names to improve readability and better suit your needs.

To rename a view, you can use either:

- Double-click directly on the view name (the selected view in this case). The name then becomes editable:



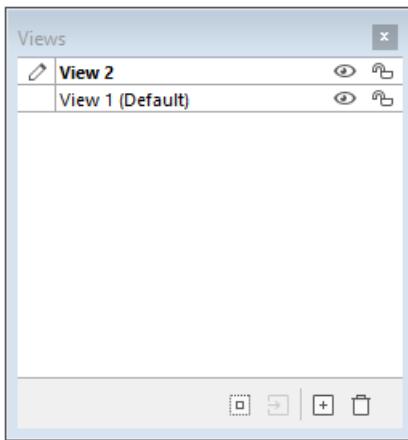
- Right-click on the view name. The name then becomes editable:



## Reordering views

You can change the display order of views by dragging/dropping them within the View palette.

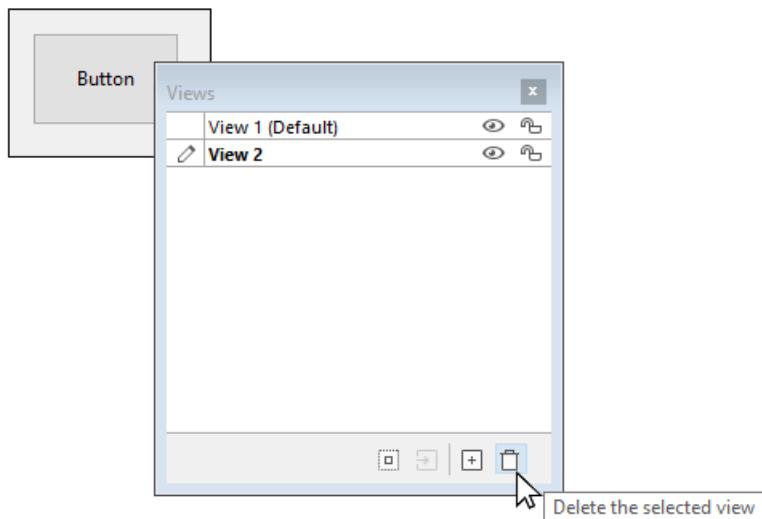
Note that the Default view does not change:



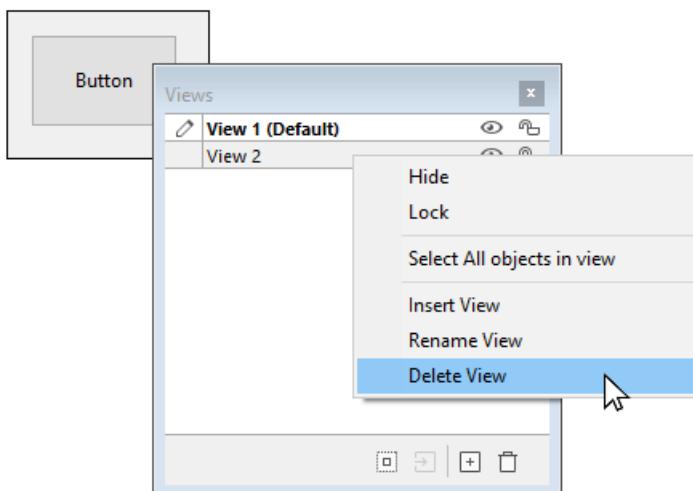
## Deleting views

To rename a view, you can use either:

- Click on the **Delete the selected view** button at the bottom of the View palette:



- Right-click on the view name, and select **Delete View**:



If a view is deleted, any objects in it are automatically moved to the Default view.

## Using views

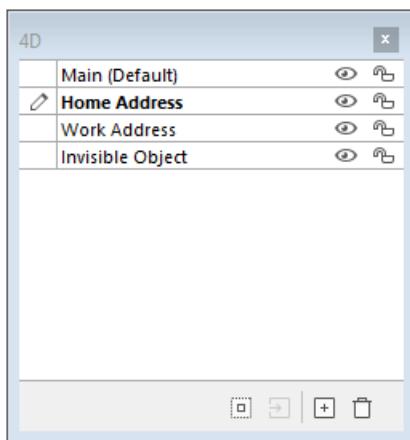
Once views are created, you can use the View palette to:

- Add object to views,
- Move objects from one view to another,
- Select all objects of the same view in a single click,
- Display or hide objects for each view,
- Lock the objects of a view.

## Adding objects to views

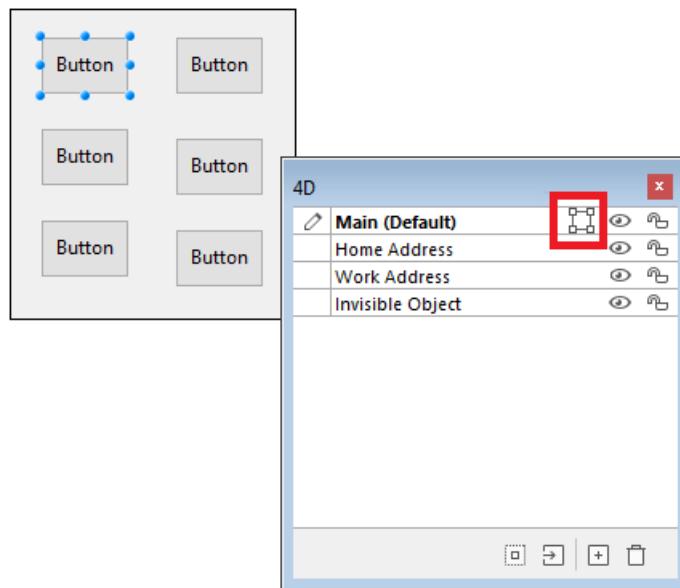
An object can only belong to a single view.

To create an object in another view, simply select the view in the View palette (prior to creating the object) by clicking its name (an Edit icon is displayed for the Current view and the name appears in bold text):



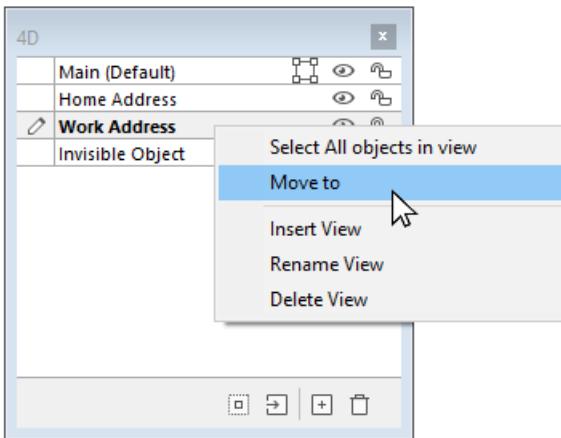
## Moving objects between views

It's also possible to move one or more objects from one view to another. In the form, select the object(s) whose view you wish to change. The view list indicates, using a symbol, the view to which the selection belongs:



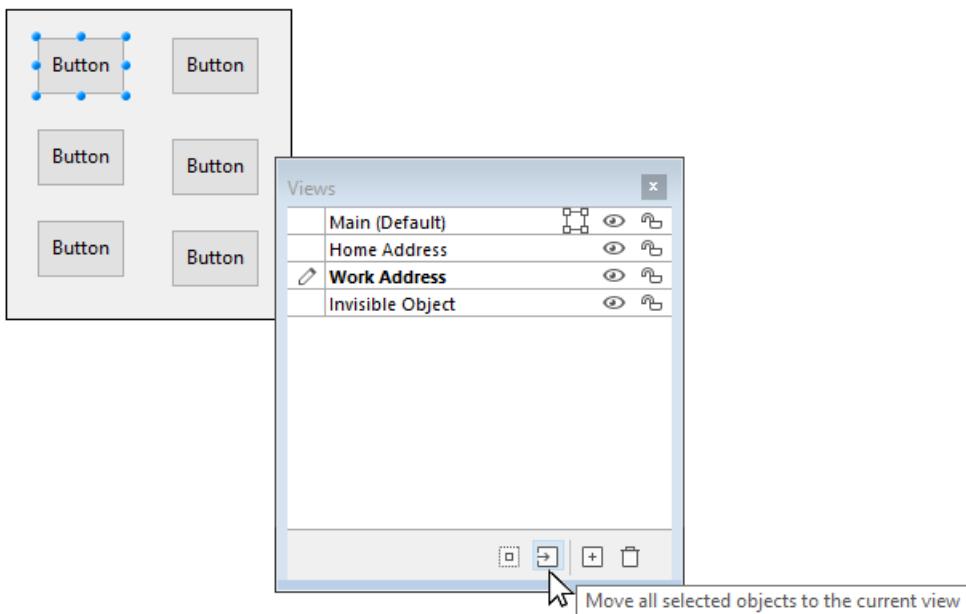
The selection can contain several objects belonging to different views.

Simply select the destination view, right-click, and select **Move to**:

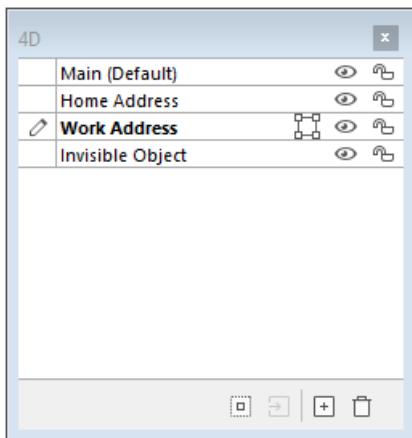


OR

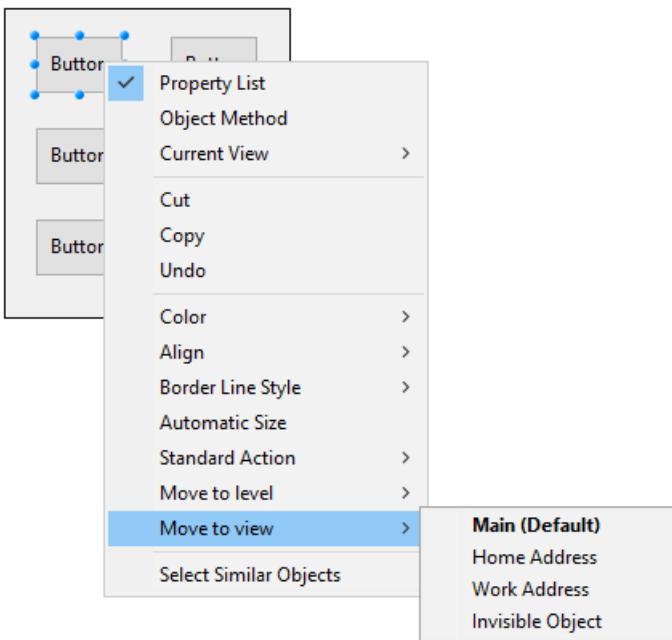
Select the destination view of the selection and click **Move to** button at the bottom of the View palette:



The selection is then placed in the new view:



You can also move an object to another view via the object's context menu. Right-click on the object, select **Move to view**, and select a view from the list of available views:

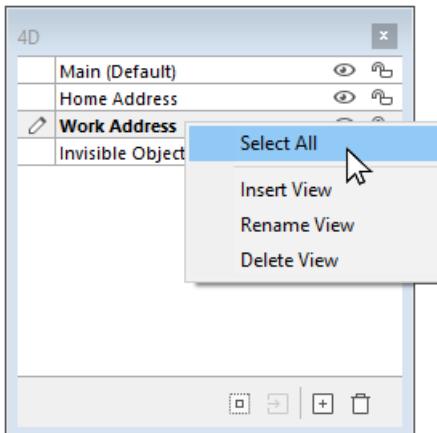


The [Current view](#) is shown in bold text.

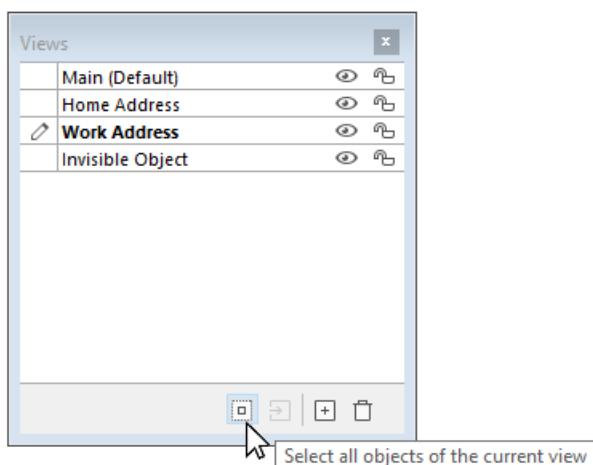
### Select all objects of a view

You can select all objects belong to the same view in the current page of the form. This function is useful for applying global changes to a set of objects.

To do this, right-click on the view in which you wish to select all the objects, click on **Select All**:



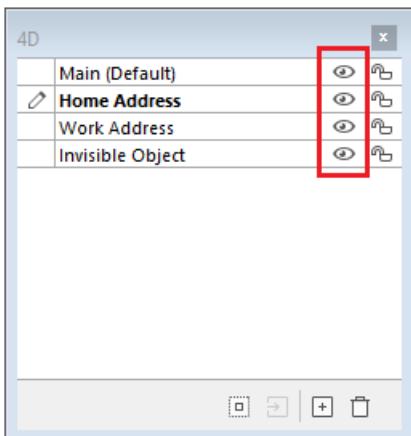
You can also use the button at the bottom of the View palette:



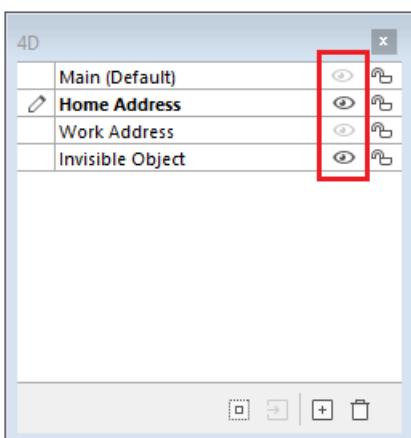
## Show or hide objects of a view

You can show or hide objects belonging to a view at any time in the form's current page. This way you can focus on certain objects when editing the form, for example.

By default, all views are shown, as indicated by the *Show/Hide* icon:



To hide a view, click the *Show/Hide* icon. It is then dimmed and objects of the corresponding view are no longer shown in the form:



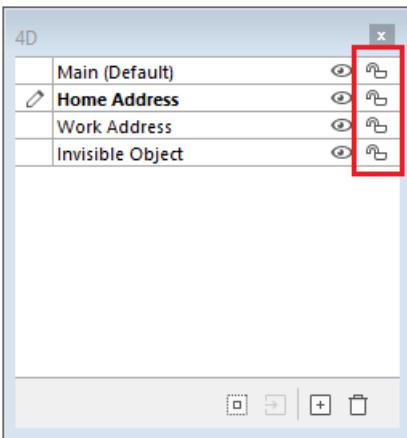
The [Current view](#) cannot be hidden.

To show a view that is hidden, simply select it or click on the *Show/Hide* icon for that view.

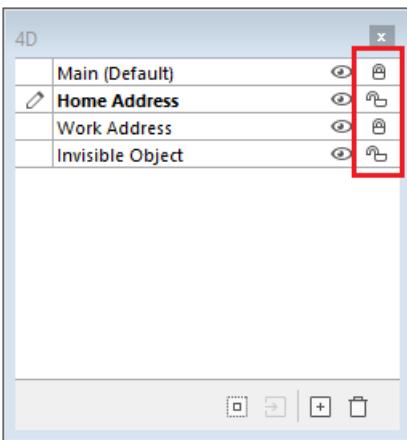
## Locking objects of a view

You can lock the objects of a view. This prevents them from being selected, changed, or deleted from the form. Once locked, an object cannot be selected by a click, a rectangle, or the **Select Similar Objects** command of the context menu. This function is useful for preventing handling errors.

By default, all views are unlocked, as indicated by the *Lock/Unlock* icon next to each view:



To lock the objects of a view, click the *Lock/Unlock* icon. The padlock is shut, which means that the view is now locked:



The [Current view](#) cannot be locked.

To unlock a view that is locked, simply select it or click on the *Lock/Unlock* icon for that view.

## Zoom

You can zoom in the current form. Switch to “Zoom” mode by clicking on the magnifying glass icon or clicking directly on the desired percentage bar (50%, 100%, 200%, 400% and 800%):



- When you click on the magnifying glass, the cursor changes into one. You can then click in the form to increase the display or hold down Shift and click to reduce the display percentage.
- When you click on a percentage bar, the display is immediately modified.

In Zoom mode, all Form editor functions remain available(\*)).

(\*) For technical reasons, it is not possible to select list box elements (headers, columns, footers) when the Form editor is in Zoom mode.

## Form Editor Macros

The 4D Form editor supports macros. A macro is a set of instructions to perform an action or a sequence of actions. When called upon, the macro will execute its instructions and automatically perform the action(s).

For example if you have a recurring report with specific formatting (e.g., certain text must appear in red and certain text must appear in green), you can create a macro to automatically set the color. You can create macros for the 4D Form editor that can:

- Create and execute 4D code
- Display dialogs
- Select form objects
- Add / delete / modify forms, form objects as well as their properties
- Modify project files (update, delete)

Macros code supports [class functions](#) and [form object properties in JSON](#) to let you define any custom feature in the Form editor.

Macros can be defined for the host project or for components within the project. Usually, you will create a macro and install it within the components you use for development.

When called, a macro overrides any previously specified behaviors.

## Hands-on example

In this short example, you'll see how to create and call a macro that adds a "Hello World!" alert button in the top left corner of your form.

1. In a `formMacros.json` file within the `Sources` folder of your project, you write:

```
{  
  "macros": {  
    "Add Hello World button": {  
      "class": "AddButton"  
    }  
  }  
}
```

2. Create a 4D class named `AddButton`.
3. Within the `AddButton` class, write the following function:

```

Function onInvoke($editor : Object) -> $result : Object
    var $btnHello : Object

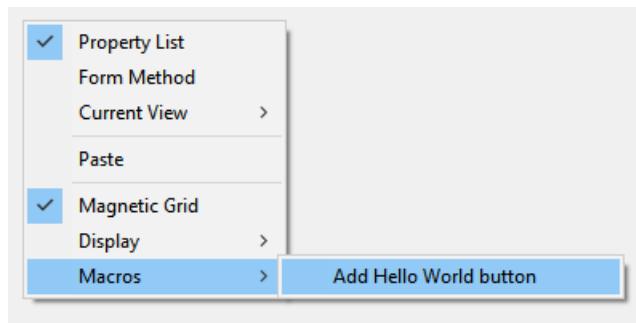
    // Create a "Hello" button
    $btnHello:=New object("type"; "button"; \
    "text"; "Hello World!"; \
    "method"; New object("source"; "ALERT(\"Hello World!\")"); \
    "events"; New collection("onClick"); \
    "width"; 120; \
    "height"; 20; \
    "top"; 0; \
    "left"; 0)

    // Add button in the current page
    $editor.editor.currentPage.objects.btnHello:=$btnHello

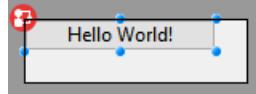
    // Select the new button in the form editor
    $editor.editor.currentSelection.clear() //unselect elements
    $editor.editor.currentSelection.push("btnHello")

    // Notify the modification to the 4D Form editor
    $result:=New object("currentSelection";
    $editor.editor.currentSelection; \
    "currentPage"; $editor.editor.currentPage)

```

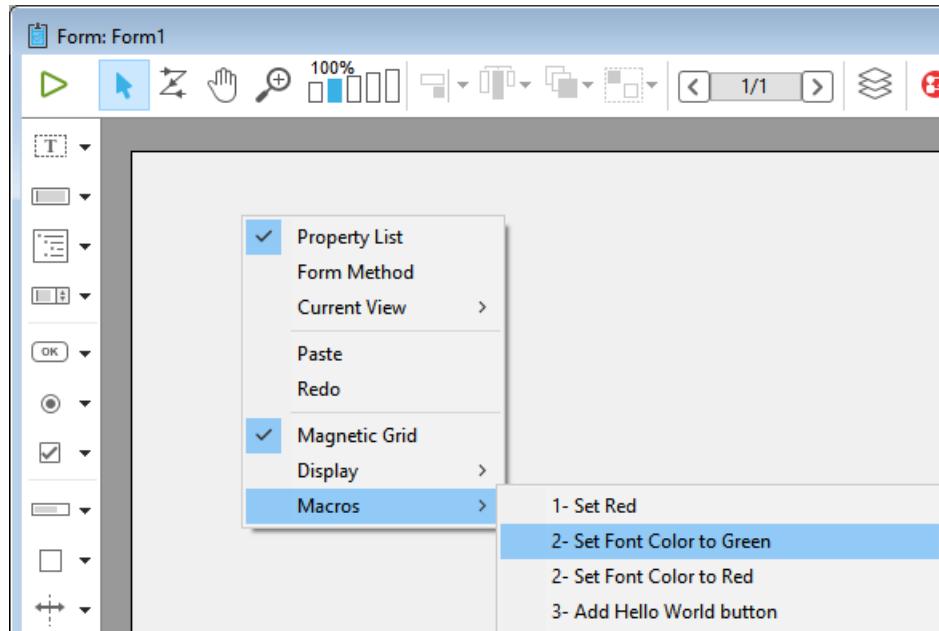


You can then call the macro:



## Calling macros in the Form editor

When macros are defined in your 4D project, you can call a macro using the contextual menu of the Form editor:



This menu is built upon the `formMacros.json` [macro definition file\(s\)](#). Macro items are sorted in alphabetical order.

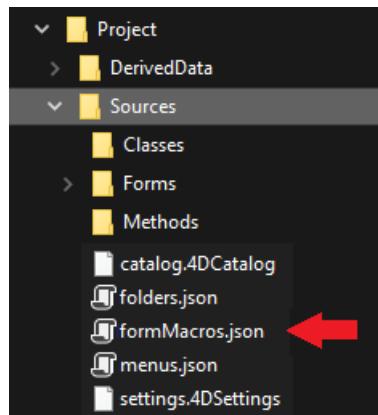
This menu can be called in an empty area or a selection in the form. Selected object are passed to `$editor.currentSelection` or `$editor.target` in the [onInvoke](#) function of the macro.

A single macro can execute several operations. If selected, the **Undo** feature of the Form editor can be used to reverse macro operations globally.

## Location of macro file

All 4D Form Editor macros are defined within a single JSON file per project or component: `FormMacros.json`.

This file must be located in the host or component's **Project > Sources** folder:



## Declaring macros

The structure of the `formMacros.json` file is the following:

```
{
  "macros": {
    <macroName>: {
      "class": <className>,
      <customProperty> : <value>
    }
  }
}
```

Here is the description of the JSON file contents:

<b>Attribute</b>	<b>Type</b>	<b>Description</b>
macros	objectlist of defined macros	
<macroName>	objectmacro definition	
class	string macro class name	
<customProperty>	any	(optional) custom value to retrieve in the constructor

Custom properties, when used, are passed to the [constructor](#) function of the macro.

## Example

```
{
  "macros": {
    "Open Macros file": {
      "class": "OpenMacro"
    },
    "Align to Right on Target Object": {
      "class": "AlignOnTarget",
      "myParam": "right"
    },
    "Align to Left on Target Object": {
      "class": "AlignOnTarget",
      "myParam": "left"
    }
  }
}
```

## Instantiating macros in 4D

Each macro you want to instantiate in your project or component must be declared as a [4D class](#).

The class name must match the name defined using the [class](#) attribute of the `formMacros.json` file.

Macros are instantiated at application startup. Consequently, if you modify the macro class structure (add a function, modify a parameter...) or the [constructor](#), you will have to restart the application to apply the changes.

## Macro Functions

Every macro class can contain a `Class constructor` and two functions: `onInvoke()` and `onError()`.

### Class constructor

#### Class constructor(\$macro : Object)

Parameter Type	Description
\$macro Object	Macro declaration object (in the <code>formMacros.json</code> file)

Macros are instantiated using a [class constructor](#) function, if it exists.

The class constructor is called once during class instantiation, which occurs at application startup.

Custom properties added to the [macro declaration](#) are returned in the parameter of the class constructor function.

## Example

In the `formMacros.json` file:

```
{
  "macros": {
    "Align to Left on Target Object": {
      "class": "AlignOnTarget",
      "myParam": "left"
    }
  }
}
```

You can write:

```
// Class "AlignOnTarget"
Class constructor($macro : Object)
  This.myParameter:=$macro.myParam //left
  ...
  
```

## onInvoke()

### onInvoke(\$editor : Object) -> \$result : Object

Parameter Type	Description
\$editor Object	Form Editor Macro Proxy object containing the form properties
\$result Object	Form Editor Macro Proxy object returning properties modified by the macro (optional)

The `onInvoke` function is automatically executed each time the macro is called.

When the function is called, it receives in the `$editor.editor` property a copy of all the elements of the form with their current values. You can then execute any operation on these properties.

Once operations are completed, if the macro results in modifying, adding, or removing objects, you can pass the resulting edited properties in `$result`. The macro processor will parse the returned properties and apply necessary operations in the form. Obviously, the less properties you return, the less time processing will require.

Here are the properties returned in the `$editor` parameter:

Property	Type	Description
\$editor.editor.form	Object	The entire form
\$editor.editor.file	File	File object of the form file
\$editor.editor.name	String	Name of the form
\$editor.editor.table	number	Table number of the form, 0 for project form
\$editor.editor.currentPageNumber	number	The number of the current page
\$editor.editor.currentPage	Object	The current page, containing all the form objects and the entry order of the page
\$editor.editor.currentSelection	Collection	Collection of names of selected objects
\$editor.editor.formProperties	Object	Properties of the current form
\$editor.editor.target	string	Name of the object under the mouse when clicked on a macro

Here are the properties that you can pass in the `$result` object if you want the macro processor to execute a modification. All properties are optional:

Property	Type	Description
currentPage	Object	currentPage including objects modified by the macro, if any
currentSelection	Collection	currentSelection if modified by the macro
formProperties	Object	formProperties if modified by the macro
editor.groups	Object	group info, if groups are modified by the macro
editor.views	Object	view info, if views are modified by the macro
editor.activeView	String	Active view name

For example, if objects of the current page and groups have been modified, you can write:

```
$result:=New object("currentPage"; $editor.editor.currentPage ; \
                     "editor"; New object("groups"; \
$editor.editor.form.editor.groups))
```

### method attribute

When handling the `method` attribute of form objects, you can define the attribute value in two ways in macros:

- Using a [string containing the method file name/path.](#)
- Using an object with the following structure:

### Property Type Description

source String method code

4D will create a file using the object name in the "objectMethods" folder with the content of `source` attribute. This feature is only available for macro code.

### \$4dId property in currentPage.objects

The `$4dId` property defines a unique ID for each object in the current page. This key is used by the macro processor to control changes in `$result.currentPage`:

- if the `$4dId` key is missing in both the form and an object in `$result`, the object is created.

- if the `$4dId` key exists in the form but is missing in `$result`, the object is deleted.
- if the `$4dId` key exists in both the form and an object in `$result`, the object is modified.

## Example

You want to define a macro function that will apply the red color and italic font style to any selected object(s).

```
Function onInvoke($editor : Object)->$result : Object
    var $name : Text

    If ($editor.editor.currentSelection.length>0)
        // Set stroke to red and style to italic for each selected
object
        For each ($name; $editor.editor.currentSelection)
            $editor.editor.currentPage.objects[$name].stroke:="red"

$editor.editor.currentPage.objects[$name].fontStyle:="italic"

        End for each

    Else
        ALERT("Please select a form object.")
    End if

    // Notify to 4D the modification
    $result:=New object("currentPage"; $editor.editor.currentPage)
```

## onError()

**onError(\$editor : Object; \$resultMacro : Object ; \$error : Collection)**

Parameter	Type	Description
\$editor	Object	Object send to <a href="#">onInvoke</a>
\$resultMacro	Object	Object returned by <a href="#">onInvoke</a>
\$error	Collection	Error stack
].errCode	Number	Error code
].message	Text	Description of the error
].componentSignatureText		Internal component signature

The `onError` function is executed when the macros processor encounters an error.

When executing a macro, if 4D encounters an error which prevents the macro from being cancelled, it does not execute the macro. It is the case for example if executing a macro would result in:

- deleting or modifying a script whose file is read-only.
- creating two objects with the same internal ID.

## Example

In a macro class definition, you can write the following generic error code:

```
Function onError($editor : Object; $resultMacro : Object; $error : Collection)
    var $obj : Object
    var $txt : Text
    $txt:=""

    For each ($obj; $error)
        $txt:=$txt+$obj.message+"\n"
    End for each

    ALERT($txt)
```

# Object libraries

You can use object libraries in your forms. An object library offers a collection of preconfigured objects that can be used in your forms by simple or copy-paste or drag-and-drop.

4D proposes two kinds of object libraries:

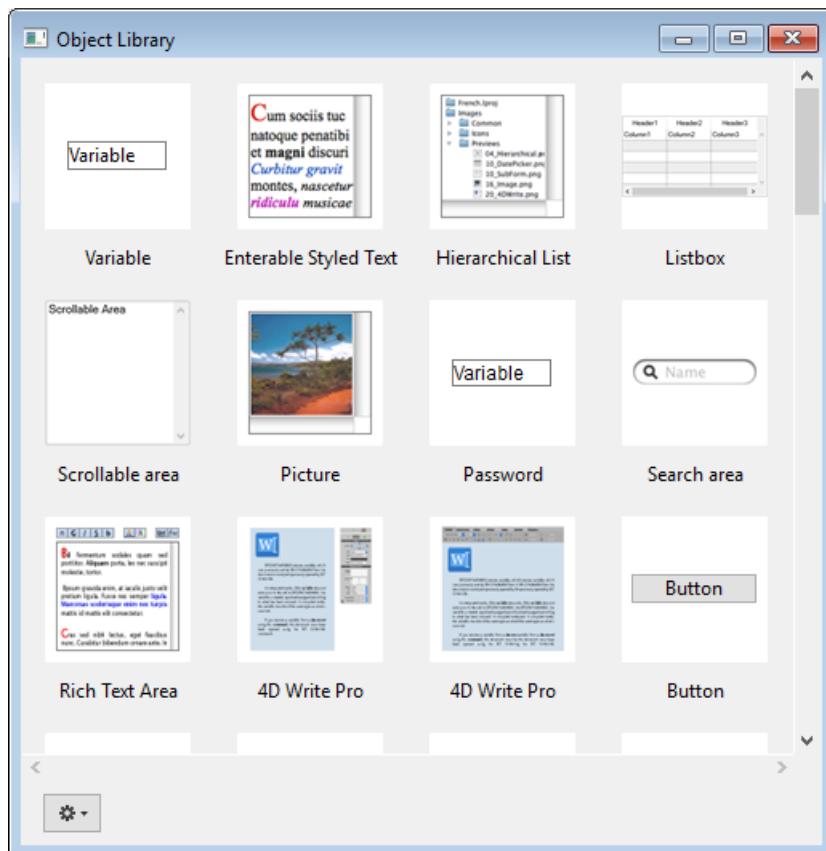
- a standard, preconfigured object library, available in all your projects.
- custom object libraries, that you can use to store your favorite form objects or full project forms.

## Using the standard object library

The standard object library is available from the Form editor: click on the last button of the toolbar:

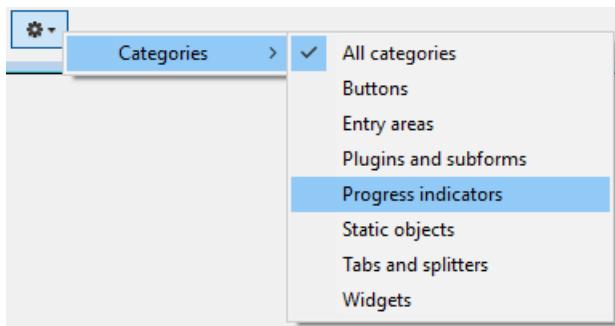


The library is displayed in a separate window:



The window has the following main features:

- **Preview area with tips:** The central area displays a preview of each object. You can hover on an object to obtain information about the object in a tip.
- You can filter the window contents by using the **Categories** menu:



- To use an object from the library to your form, you can either:
  - right-click on an object and select \*\*Copy\*\* in the contextual menu
  - or drag and drop the object from the libraryThe object is then added to the form.

This library is read-only. If you want to edit default objects or create your own library of preconfigured objects or project forms, you need to create a custom object library (see below).

All objects proposed in the standard object library are described on [this section on doc.4d.com](#).

## Creating and using custom object libraries

You can create and use custom object libraries in 4D. A custom object library is a 4D project where you can store your favorite objects (buttons, texts, pictures, etc.) You can then reuse these objects in different forms and different projects.

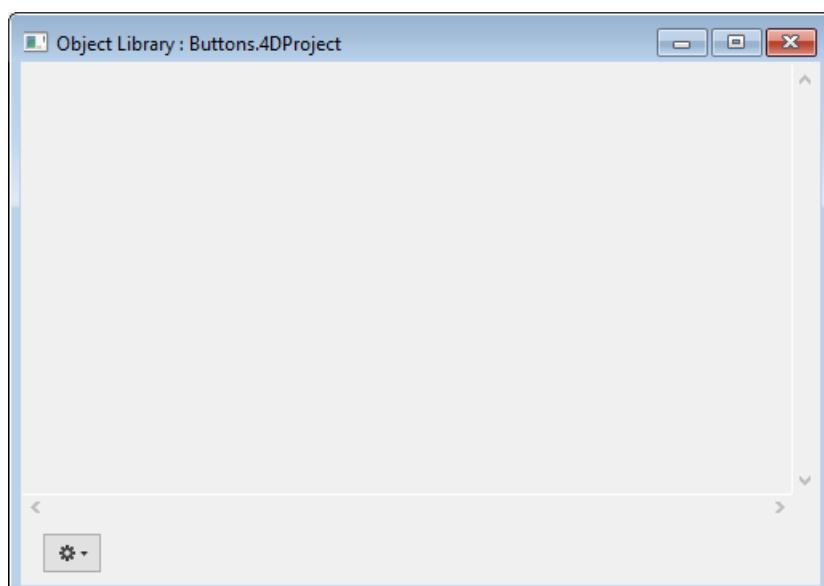
Objects are stored with all their properties, including their object methods. Libraries are put together and used by simple drag-and-drop or copy-paste operations.

Using libraries, you can build form object backgrounds grouped by graphic families, by behavior, etc.

### Creating an object library

To create an object library, select **New>Object Library...** from the 4D **File** menu or tool bar. A standard save file dialog box appears, which allows you to choose the name and the location of the object library.

Once you validate the dialog box, 4D creates a new object library on your disk and displays its window (empty by default).



You can create as many libraries as desired per project. A library created and built under macOS can be used under Windows and vice-versa.

## Opening an object library

A given object library can only be opened by one project at a time. However, several different libraries can be opened in the same project.

To open a custom object library, select **Open>Object Library...** command in the 4D **File** menu or tool bar. A standard open file dialog box appears, which allows you to select the object library to open. You can select the following file types:

- **.4dproject**
- **.4dz**

In fact, custom object libraries are regular 4D projects. Only the following parts of a project are exposed when it is opened as library:

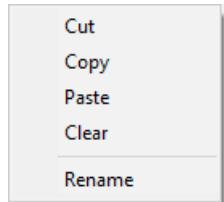
- project forms
- form pages 1

## Building an object library

Objects are placed in an object library using drag-and-drop or a cut-copy-paste operation. They can come from either a form or another object library (including the [standard library](#)). No link is kept with the original object: if the original is modified, the copied object is not affected.

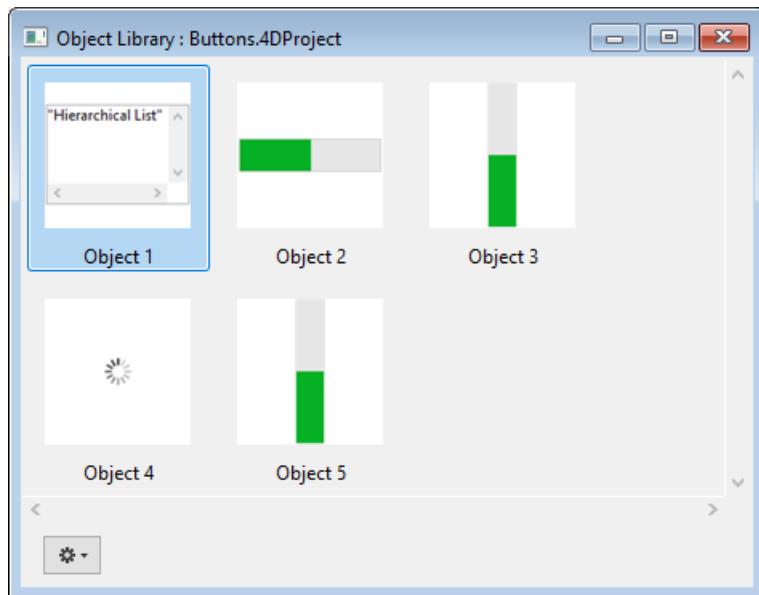
In order to be able to drag and drop objects from forms to object libraries, you must make sure the **Start drag and drop** option in the 4D Preferences is selected.

Basic operations are available in the context menu or the options menu of the window:



- **Cut** or **Copy** to the pasteboard
- **Paste** an object from the pasteboard
- **Clear** - deletes the object from the library
- **Rename** - a dialog box appears allowing you to rename the item. Note that object names must be unique in a library.

You can place individual objects (including subforms) or sets of objects in an object library. Each object or set is grouped into a single item:



An object library can contain up to 32,000 items.

Objects are copied with all their properties, both graphic and functional, including their methods. These properties are kept in full when the item is copied into a form or another library.

### Dependent objects

Using copy-paste or drag-and-drop with certain library objects also causes their dependent objects to be copied. For example, copying a button will cause the object method that may be attached to be copied as well. These dependent objects cannot be copied or dragged and dropped directly.

The following is a list of dependent objects that will be pasted into the library at the same time as the main object that uses them (when applicable):

- Lists
- Formats/Filters
- Pictures
- Help Tips (linked to a field)
- Object methods

# Form JSON property list

This page provides a comprehensive list of all form properties, sorted by their JSON name. Click on a property name to access its detailed description.

In the "Form Properties" chapter, properties are sorted according to their names and themes in the Property List.

[b](#) - [c](#) - [d](#) - [e](#) - [f](#) - [h](#) - [i](#) - [m](#) - [p](#) - [r](#) - [s](#) - [w](#)

Property	Description	Possible Values
<b>b</b>		
<a href="#">bottomMargin</a>	Vertical margin value (in pixels)	minimum: 0
<b>c</b>		
<a href="#">colorScheme</a>	Color scheme for the form	"dark", "light"
<a href="#">css</a>	CSS file(s) used by the form	CSS file path(s) provided as a string, a collection of strings, or a collection of objects with "path" and "media" properties
<b>d</b>		
<a href="#">destination</a>	Form type	"detailScreen", "listScreen", "detailPrinter", "listPrinter"
<b>e</b>		
<a href="#">entryOrder</a>	The order in which active objects are selected when the <b>Tab</b> or the <b>Carriage return</b> key is used in an input form	Collection of 4D Form object names
<a href="#">events</a>	List of all events selected for the object or form	Collection of event names, e.g. ["onClick", "onDataChange"...].
<b>f</b>		
<a href="#">formSizeAnchor</a>	Name of the object whose position determines the size of the form. (minimum length: 1)	Name of a 4D object
<b>h</b>		
<a href="#">height</a>	Height of the form	minimum: 0
<b>i</b>		
<a href="#">inheritedForm</a>	Designates the form to inherit	Name (string) of table or project form OR a POSIX path (string) to a .json file describing the form OR an object describing the form
<a href="#">inheritedFormTable</a>	Designates the table an inherited form will use	A table name or number
<b>m</b>		
<a href="#">markerBody</a>	Detail marker position	minimum: 0
<a href="#">markerBreak</a>	Break marker position(s)	minimum: 0
<a href="#">markerFooter</a>	Footer marker position	minimum: 0
<a href="#">markerHeader</a>	Header marker position(s)	integer minimum: 0; integer array minimum: 0
<a href="#">rememberFormSettings</a>	Saves the form parameters when the form window is	true, false

<u>Property</u>	<u>Description</u>	<u>Possible Values</u>
<u>menuBar</u>	Menu bar to associate to the form	Name of a valid menu bar
<u>method</u>	A project method name.	The name of an existing project method
<b>p</b>		
<u>pages</u>	Collection of pages (each page is an object) object	Page objects
<u>pageFormat</u>		Available print properties
<b>r</b>		
<u>rightMargin</u>	Horizontal margin value (in pixels)	minimum: 0
<b>s</b>		
<u>shared</u>	Specifies if a form can be used as a subform	true, false
<b>w</b>		
<u>width</u>	Width of the form	minimum: 0
<u>windowMaxHeight</u>	Form window's largest allowable height	minimum: 0
<u>windowMaxWidth</u>	Form window's largest allowable width	minimum: 0
<u>windowMinHeight</u>	Form window's smallest allowable height	minimum: 0
<u>windowMinWidth</u>	Form window's smallest allowable width	minimum: 0
<u>windowSizingX</u>	Form window's vertical sizing "fixed", "variable"	
<u>windowSizingY</u>	Form window's horizontal sizing	"fixed", "variable"
<u>windowTitle</u>	Designates a form window's title	A name for the form window

## Action

### Method

Reference of a method attached to the form. You can use a form method to manage data and objects, but it is generally simpler and more efficient to use an object method for these purposes. See [Specialized methods](#).

You do not call a form method—4D calls it automatically when an event involves the form to which the method is attached.

Several types of method references are supported:

- a standard project method file path, i.e. that uses the following pattern:

`method.4dm`

This type of reference indicates that the method file is located at the default location ("sources/{TableForms/*numTable*} | {Forms}/*formName*/"). In this case, 4D automatically handles the form method when operations are executed on the form (renaming, duplication, copy/paste...)

- a project method name: name of an existing project method without file extension, i.e.: `myMethod` In this case, 4D does not provide automatic support for form operations.
- a custom method file path including the .4dm extension, e.g.:  
`MyMethods/myFormMethod.4dm` You can also use a filesystem:  
`/RESOURCES/Forms/FormMethod.4dm` In this case, 4D does not provide automatic support for object operations.

### JSON Grammar

Name	Data Type	Possible Values
methodtext	Form method standard or custom file path, or project method name	

# Form Properties

## Color Scheme

Color scheme property is only applied on macOS.

This property defines the color scheme for the form. By default when the property is not set, the value for a color scheme is **inherited** (the form uses the scheme defined at the [application level](#)). This can be changed for the form to one of the following two options:

- dark - light text on a dark background
- light - dark text on a light background

A defined color scheme can not be overridden by a CSS.

## JSON Grammar

Name	Data Type	Possible Values
colorScheme	string	"dark", "light"

## CSS

This property allows you to load specific CSS file(s) for the form.

A CSS file defined at the form level will override default style sheet(s). For more information, please refer to [Style sheets](#) page.

## JSON Grammar

Name	Data Type	Possible Values
css	string or collection	CSS file path(s) provided as: <ul style="list-style-type: none"><li>• a string (a file for both platforms)</li><li>• a collection of strings (a list of files for both platform)</li><li>• a collection of {"path":string;"media":"mac"   "win"} objects</li></ul>

## Pages

Each form has is made of at least two pages:

- a page 0 (background page)
- a page 1 (main page)

For more information, please refer to [Form pages](#).

## JSON Grammar

Name	Data Type	Possible Values
pages	collection	Collection of pages (each page is an object, page 0 is the first element)

## Form Name

This property is the name of the form itself and is used to refer to the form by name using the 4D language. The form name must comply with the [rules specified for identifiers](#) in 4D.

## JSON Grammar

The form name is defined by the name of the folder that contains the form.4Dform file. See [project architecture](#) for more information.

## Form Type

The form type, *i.e.* its destination, defines the features that will be available to the form. For example, [markers](#) can only be set for list (output) table forms.

Each table in a database generally has at least two table forms. One for listing records on-screen and the other for displaying one record at a time (used for data entry and modifications):

- Output form - the *output form* or *list form* displays a list of records, with a single line per record. The results of queries are shown in an output form and users can double-click a line to display the input form for that record.

ID :	name :
1	Friends
3	Work
4	Personal
5	Family

- Input form - used for data entry. It displays a single record per screen and typically has buttons for saving and canceling modifications to the record and for navigating from record to record (*i.e.*, First Record, Last Record, Previous Record, Next Record).

Category	ID :	name :
Friends	136	

Supported types depend on the form category:

<b>Form Type</b>	<b>JSON grammar</b>	<b>Description</b>	<b>Supported with</b>
Detail Form	detailScreen	A display form for data entry and modification	Project forms - Table forms
Detail Form for Printing	detailPrinter	A printed report with one page per record, such as an invoice	Project forms - Table forms
List Form	listScreen	A form for listing records on the screen	Table forms
List Form for Printing	listPrinter	A printed report that lists records	Table forms
None	<i>no destination</i>	A form with no specific feature	Project forms - Table forms

## JSON Grammar

<b>Name</b>	<b>Data Type</b>	<b>Possible Values</b>
destinationString		"detailScreen", "listScreen", "detailPrinter", "listPrinter"

## Inherited Form Name

This property designates the [form to inherit](#) in the current form.

To inherit from a table form, set the table in the [Inherited Form Table](#) property.

To remove inheritance, select `\<None>` in the Property List (or " " in JSON).

## JSON Grammar

<b>Name</b>	<b>Data Type</b>	<b>Possible Values</b>
inheritedFormString		Name of table or project form OR a POSIX path to a .json file describing the form OR an object describing the form

## Inherited Form Table

This property specifies the database table from which to [inherit a form](#) in the current form.

Set to `\<None>` in the Property List (or " " in JSON) to inherit from a project form.

## JSON Grammar

<b>Name</b>	<b>Data Type</b>	<b>Possible Values</b>
inheritedFormTableName	string or number	table name or table number

## Published as Subform

For a component form to be selected as a [subform](#) in a host application, it must have been explicitly shared. When this property is selected, the form will be published in the host application.

Only project forms can be specified as published subforms.

## JSON Grammar

## Name Data Type Possible Values

shared boolean true, false

## Save Geometry

When the option is used, if the window is opened using the `Open form window` command with the `*` parameter, several form parameters are automatically saved by 4D when the window is closed, regardless of how they were modified during the session:

- the current page,
- the position, size and visibility of each form object (including the size and visibility of list box columns).

This option does not take into account objects generated using the `OBJECT DUPLICATE` command. In order for a user to recover their environment when using this command, the developer must repeat the sequence of creation, definition and positioning of the objects.

When this option is selected, the [Save Value](#) option is available for certain objects.

## JSON Grammar

### Name Data Type Possible Values

memorizeGeometry boolean true, false

## See also

[Save Value](#)

## Window Title

The window title is used when the form is opened using the `Open form window` and `Open window` 4D commands in Application environment. The window title appears in the Title bar of the window.

You can use dynamic references to set the window titles for forms, i.e.:

- A standard XLIFF reference stored in the Resources folder.
- A table or field label: The syntax to apply is `<? [TableName] FieldNum >` or `<? [TableName] FieldName >`.
- A variable or a field: The syntax to apply is `\<VariableName>` or `< [TableName] FieldName >`. The current value of the field or variable will be displayed in the window title.

The number of characters for a window title is limited to 31.

## JSON Grammar

### Name Data Type

windowTitle string The name of the window as plain text or as a reference

### Possible Values

## Form Size

4D lets you set the size of both the form and the [window](#). These properties are interdependent and your application interface results from their interaction.

Size options depend on the value of the **Size based on** option.

### Size based on

- **Automatic Size:** The size of the form will be that necessary to display all the objects, to which will be added the margin values (in pixels) entered in the [Hor Margin](#) and [Vert Margin](#) fields.

You can choose this option when you want to use active objects placed in an offscreen area (*i.e.*, outside the bounding rectangle of the window) with an automatic size window. Thanks to this option, the presence of these objects will not modify the size of the window.

- **Set Size:** The size of the form will be based on what you enter (in pixels) in the [Width](#) and [Height](#) fields.
- `<object name>`: The size of the form will be based on the position of the selected form object. For example, if you choose an object that is placed in the bottom-right part of the area to be displayed, the form size will consist of a rectangle whose upper left corner will be the origin of the form and the lower right corner will correspond to that of the selected object, plus any margin values.

For output forms, only the [Hor margin](#) or [Width](#) fields are available.

### JSON Grammar

Name	Data Type	Possible Values
formSizeAnchor	string	Name of object to use to defined the size of the form

## Height

Height of the form (in pixels) when the [form size](#) is **Set size**.

### JSON Grammar

Name	Data Type	Possible Values
height	number	integer value

## Hor. Margin

Value to add (in pixels) to the right margin of the form when the [form size](#) is **Automatic size** or `\<object name>`

This value also determines the right-hand margins of forms used in the Label editor.

### JSON Grammar

Name	Data Type	Possible Values
rightMargin	number	integer value

## Vert. Margin

Value to add (in pixels) to the bottom margin of the form when the [form size](#) is **Automatic size** or `\<object name>`.

This value also determines the top margins of forms used in the Label editor.

### JSON Grammar

Name	Data Type	Possible Values
bottomMargin	number	integer value

## Width

Width of the form (in pixels) when the [form size](#) is **Set size**.

### JSON Grammar

Name	Data Type	Possible Values
width	number	integer value

## Markers

These properties let you specify the precise location of markers on the vertical ruler of a **table form**. Markers are mainly used in output forms. They control the information that is listed and set header, breaks, detail and footer areas of the form. Any object that placed in these areas is displayed or printed at the appropriate location.

Whenever any form is used for output, either for screen display or printing, the output marker lines take effect and the areas display or print at designated locations. The markers also take effect when a form is used as the List form in a subform area. However, they have no effect when a form is used for input.

Methods that are associated with objects in these areas are executed when the areas are printed or displayed as long as the appropriate events have been activated. For example, a object method placed in the Header area is executed when the `On Header` event takes place.

## Form Break

Form Break areas are displayed once at the end of the list of records and are printed once after the records have been printed in a report.

The Break area is defined as the area between the Detail control line and the Break control line. There can be [several Break areas](#) in your report.

You can make Break areas smaller or larger. You can use a Break area to display information that is not part of the records (instructions, current date, current time, etc.), or to display a line or other graphic element that concludes the screen display. In a printed report, you can use a Break area for calculating and printing subtotals and other summary calculations.

### JSON Grammar

Name	Data Type	Possible Values
markerBreak	integer   integer collection	Break marker position or collection of break marker positions in pixels. Minimum value: 0

## Form Detail

The form Detail area is displayed on the screen and printed once for each record in a report. The Detail area is defined as the area between the Header control line and the Detail control line.

You can make the Detail area smaller or larger. Whatever you place in the Detail area is displayed or printed once for each record. Most often you place fields or variables in the Detail area so that the information in each record is displayed or printed, but you can place other elements in the Detail area as well.

### JSON Grammar

Name	Data Type	Possible Values
markerBody	integer	Detail marker position. Minimum: 0

## Form Footer

The Form Footer area is displayed on screen under the list of records. It is always printed at the bottom of every page of a report. The Footer area is defined as the area between the Break control line and the Footer control line.

You make the Footer area smaller or larger.

You can use the Footer area to print graphics, page numbers, the current date, or any text you want at the bottom of each page of a report. For output forms designed for use on screen, the Footer area typically contains buttons that give the user options such as doing a search or sort, printing records, or putting away the current report. Active objects are accepted.

### JSON Grammar

Name	Data Type	Possible Values
markerFooter	integer	minimum: 0

## Form Header

The form Header area is displayed at the top of each screen and is printed at the top of each page of a report. The Header area is defined as the area above the Header control line.

You can make the Header area smaller or larger. You can use the Header area for column names, for instructions, additional information, or even a graphic such as a company logo or a decorative pattern.

You can also place and use active objects in the Header area of output forms displayed as subforms, in the records display window or using the `DISPLAY SELECTION` and `MODIFY SELECTION` commands. The following active objects can be inserted:

- Buttons, picture buttons,
- Combo boxes, drop-down lists, picture pop-up menus,
- hierarchical lists, list boxes
- Radio buttons, check boxes, 3D check boxes,
- Progress indicators, rulers, steppers, spinners.

Standard actions such as `Add Subrecord`, `Cancel` (lists displayed using `DISPLAY SELECTION` and `MODIFY SELECTION`) or `Automatic splitter` can be assigned to the inserted buttons. The following events apply to the active objects you insert in the Header area: `On Load`, `On Clicked`, `On Header`, `On Printing Footer`, `On Double Clicked`, `On Drop`, `On Drag Over`, `On Unload`. Keep in mind that the form method is called with the `On Header` event after calling the object methods of the area.

The form can contain [additional header areas](#) to be associated with additional breaks. A level 1 Header is printed just before the records grouped by the first sorted field are printed.

### JSON Grammar

Name	Data Type	Possible Values
markerHeader	integer   integer collection	Header marker position or collection of header marker positions in pixels. Minimum value: 0

## Additional areas

You can create additional Break areas and Header areas for a report. These additional areas allow you to print subtotals and other calculations in a report and to display other information effectively.

Additional areas are defined when you use a collection of positions in the [Form Break](#) and [Form Header](#) properties.

In the 4D Form editor, you create additional control lines by holding down the **Alt** key while clicking the appropriate control marker.

A form always starts with a Header, Detail, Break level 0, and Footer areas.

Break at level 0 zero takes in all the records; it occurs after all the records are printed. Additional Break areas can be added, i.e. a Break level 1, Break level 2, etc.

A Break level 1 occurs after the records grouped by the first sorted field are printed.

<b>Label</b>	<b>Description</b>	<b>Prints after groups created by</b>
--------------	--------------------	---------------------------------------

Form Break 1	Break at level 1	First sorted field
Form Break 2	Break at level 2	Second sorted field
Form Break 3	Break at level 3	Third sorted field

Additional Header areas are associated with Breaks. A level 1 Header is printed just before the records grouped by the first sorted field are printed.

<b>Label</b>	<b>Description</b>	<b>Prints after groups created by</b>
--------------	--------------------	---------------------------------------

Form Header 1	Header at level 1	First sorted field
Form Header 2	Header at level 2	Second sorted field
Form Header 3	Header at level 3	Third sorted field

If you use the **Subtotal** function to initiate Break processing, you should create a Break area for every level of Break that will be generated by the sort order, minus one. If you do not need anything printed in one of the Break areas, you can reduce its size to nothing by placing its marker on top of another control line. If you have more sort levels than Break areas, the last Break area will be repeated during printing.

## **Menu**

### **Associated Menu Bar**

When a menu bar is associated to a form, it is added to the right of the current menu bar when the form is displayed in Application environment.

The selection of a menu command causes an `On Menu Selected` event to be sent to the form method; you can then use the `Menu selected` command to test the selected menu.

If the menu bar of the form is identical to the current menu bar, it is not added.

The form menu bar will operate for both input and output forms.

### **JSON Grammar**

<b>Name</b>	<b>Data Type</b>	<b>Possible Values</b>
menuBarString	Name of a menu bar	

## Print

### Settings

Allows defining specific print settings for the form. This feature is useful to view printing page limits in the form editor.

**Compatibility:** Even if these settings are taken into account when the form is printed in Application mode, it is discouraged to rely on this feature to store print settings for the form, due to limitations regarding the platform and driver dependency. It is highly recommended to use the 4D commands `Print settings to BLOB/BLOB to print settings` which are more powerful.

You can modify the following print settings:

- Paper format
- Paper orientation
- Page scaling

Available options depend on the system configuration.

### JSON Grammar

Name	Data Type	Possible Values
pageFormat	object	Available print properties: <code>paperName</code> , <code>paperWidth</code> , <code>paperHeight</code> , <code>orientation</code> , <code>scale</code>
paperName	string	"A4", "US Letter"...
paperWidth	string	Used if a paper named <code>paperName</code> was not found. Requires unit suffix: <code>pt</code> , <code>in</code> , <code>mm</code> , <code>cm</code> .
paperHeight	string	Used if a paper named <code>paperName</code> was not found. Requires unit suffix: <code>pt</code> , <code>in</code> , <code>mm</code> , <code>cm</code> .
orientation	string	"landscape" (default is "portrait")
scale	number	minimum: 0

## Window Size

### Fixed Height

If you select this option, the window height will be locked and it will not be possible for the user to resize it.

If this option is not selected, the width of the form window can be modified. In this case, the [Minimum Height and Maximum Height](#) properties can be used to determine the resizing limits.

#### JSON Grammar

Name	Data Type	Possible Values
windowSizingY	string	"fixed", "variable"

### Fixed Width

If you select this option, the window width will be locked and it will not be possible for the user to resize it.

If this option is not selected, the width of the form window can be modified. In this case, the [Minimum Width and Maximum Width](#) properties can be used to determine the resizing limits.

#### JSON Grammar

Name	Data Type	Possible Values
windowSizingX	string	"fixed", "variable"

### Maximum Height, Minimum Height

Maximum and minimum height (in pixels) of a resizable form window if the [Fixed Height](#) option is not set.

#### JSON Grammar

Name	Data Type	Possible Values
windowMinHeight	number	integer value
windowMaxHeight	number	integer value

### Maximum Width, Minimum Width

Maximum and minimum width (in pixels) of a resizable form window if the [Fixed Width](#) option is not set.

#### JSON Grammar

Name	Data Type	Possible Values
windowMinWidth	number	integer value
windowMaxWidth	number	integer value



# Form Objects

You build and customize your application forms by manipulating the objects on them. You can add objects, reposition objects, set object properties, enforce business rules by specifying data entry constraints, or write object methods that run automatically when the object is used.

## Active and static objects

4D forms support a large number of built-in **active** and **static** objects:

- **active objects** perform a database task or an interface function. Fields are active objects. Other active objects — enterable objects (variables), combo boxes, drop-down lists, picture buttons, and so on — store data temporarily in memory or perform some action such as opening a dialog box, printing a report, or starting a background process.
- **static objects** are generally used for setting the appearance of the form and its labels as well as for the graphic interface. Static objects do not have associated variables like active objects. However, you can insert dynamic objects into static objects.

## Handling form objects

You can add or modify 4D form objects in the following ways:

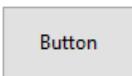
- **Form Editor:** Drag an object from the Form Editor toolbar onto the form. Then use the Property List to specify the object's properties.  
See the [Building Forms](#) chapter for more information.
- **4D language:** Commands from the [Objects \(Forms\)](#) theme such as `OBJECT`, `DUPLICATE` or `OBJECT SET FONT STYLE` allow to create and define form objects.
- **JSON code in dynamic forms:** Define the properties using JSON. Use the `type` property to define the object type, then set its available properties. See the [Dynamic Forms](#) page for information.

Example for a button object:

```
```
{
    "type": "button",
    "style": "bevel",
    "text": "OK",
    "action": "Cancel",
    "left": 60,
    "top": 160,
    "width": 100,
    "height": 20
}
```

## Button

A button is an active object that can be assigned an action (e.g., a database task or an interface function) to perform when a user clicks on it.



Buttons can fulfill a variety of roles, depending on their style and the action assigned to it. For example, buttons could lead a user through a questionnaire or form to complete, or to make choices. Depending on its settings, a button may be designed to be clicked only once and execute a command, while others may require the user to click more than once to receive the desired result.

## Handling buttons

The actions assigned to buttons can originate from predefined [standard actions](#) or from custom object methods. Examples of typical actions include letting the user accept, cancel, or delete records, copy or paste data, move from page to page in a multi-page form, open, delete, or add records in a subform, handle font attributes in text areas, etc.

Buttons with standard actions are dimmed when appropriate during form execution. For example, if the first record of a table is displayed, a button with the `firstRecord` standard action would appear dimmed.

If you want a button to perform an action that's not available as a standard action, leave the standard action field empty and write an object method to specify the button's action. For more information about object methods and how to create and associate them, see [Using object methods](#). Normally, you would activate the `On Clicked` event and the method would run only when the button is clicked. You can associate a method with any button.

The [variable](#) associated with a button is automatically set to **0** when the form is executed for the first time in Design or Application mode. When the user clicks a button, its variable is set to **1**.

A button can be assigned both a standard action and a method. In this case, if the button is not disabled by the standard action, the method is executed before the standard action.

## Button Styles

Button styles control a button's general appearance as well as its available properties. It is possible to apply different predefined styles to buttons or to associate pop-up menus with them. A great number of variations can be obtained by combining these properties / behaviors.

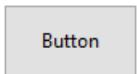
With the exception of the [available properties](#), many button objects are *structurally* identical. The difference is in the processing of their associated variables.

4D provides buttons in the following predefined styles:

### Regular

The Regular button style is a standard system button (i.e., a rectangle with a

descriptive label) which executes code when a user clicks on it.



By default, the Regular style has a light gray background with a label in the center. When the cursor hovers over the Regular button style, the border and background color change to demonstrate that it has the focus. In addition to initiating code execution, the Regular button style mimics a mechanical button by quickly changing background color when being clicked.

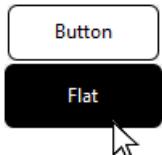
### JSON Example:

```
"myButton": {  
    "type": "button",    //define the type of object  
    "style": "regular", //define the style of the button  
    "defaultButton": "true" //define button as the default choice  
    "text": "OK",      //text to appear on the button  
    "action": "Cancel", //action to be performed  
    "left": 60,        //left position on the form  
    "top": 160,        //top position on the form  
    "width": 100,       //width of the button  
    "height": 20 //height of the button  
}
```

Only the Regular and Flat styles offer the [Default Button](#) property.

### Flat

The Flat button style is a standard system button (*i.e.*, a rectangle with a descriptive label) which executes code when a user clicks on it.



By default, the Flat style has a white background with a label in the center, rounded corners, and a minimalist appearance. The Flat button style's graphic nature is particularly useful for forms that will be printed.

### JSON Example:

```
"myButton": {  
    "type": "button",  
    "style": "flat",  
    "defaultButton": "true"  
    "text": "OK",  
    "action": "Cancel",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

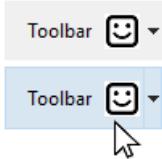
Only the Regular and Flat styles offer the [Default Button](#) property.

### Toolbar

The Toolbar button style is primarily intended for integration in a toolbar. It includes the option to add a pop-up menu (indicated by an inverted triangle) which is generally used to display additional choices for the user to select.

By default, the Toolbar style has a transparent background with a label in the center. The appearance of the button can be different when the cursor hovers over it depending on the OS:

- Windows - the button is highlighted when it uses the “With Pop-up Menu” property, a triangle is displayed to the right and in the center of the button.



- macOS - the highlight of the button never appears. When it uses the “With Pop-up Menu” property, a triangle is displayed to the right and at the bottom of the button.

### JSON Example:

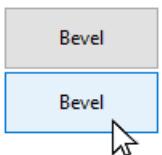
```
"myButton": {  
    "type": "button",  
    "style": "toolbar",  
    "text": "OK",  
    "popupPlacement": "separated",  
    "action": "Cancel",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

### Bevel

The Bevel button style combines the appearance of the [Regular](#) (*i.e.*, a rectangle with a descriptive label) style with the [Toolbar](#) style's pop-up menu property option.

By default, the Bevel style has a light gray background with a label in the center. The appearance of the button can be different when the cursor hovers over it depending on the OS:

- Windows - the button is highlighted. When it uses the “With Pop-up Menu” property, a triangle is displayed to the right and in the center of the button.



- macOS - the highlight of the button never appears. When it uses the “With Pop-up Menu” property, a triangle is displayed to the right and at the bottom of the button.

### JSON Example:

```

"myButton": {
    "type": "button",
    "style": "bevel",
    "text": "OK",
    "popupPlacement": "linked"
    "action": "Cancel",
    "left": 60,
    "top": 160,
    "width": 100,
    "height": 20
}

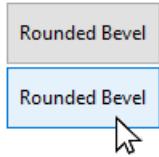
```

## Rounded Bevel

The Rounded Bevel button style is nearly identical to the [Bevel](#) style except, depending on the OS, the corners of the button may be rounded. As with the Bevel style, the Rounded Bevel style combines the appearance of the [Regular](#) style with the [Toolbar](#) style's pop-up menu property option.

By default, the Rounded Bevel style has a light gray background with a label in the center. The appearance of the button can be different when the cursor hovers over it depending on the OS:

- *Windows* - the button is identical to the Bevel style. When it uses the "With Pop-up Menu" property, a triangle is displayed to the right and in the center of the button.



- *macOS* - the corners of the button are rounded. When it uses the "With Pop-up Menu" property, a triangle is displayed to the right and at the bottom of the button.

## JSON Example:

```

"myButton": {
    "type": "button",
    "style": "roundedBevel",
    "text": "OK",
    "popupPlacement": "none" /
    "action": "Cancel",
    "left": 60,
    "top": 160,
    "width": 100,
    "height": 20
}

```

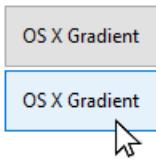
## OS X Gradient

The OS X Gradient button style is nearly identical to the [Bevel](#) style. As with the Bevel style, the OS X Gradient style combines the appearance of the [Regular](#) style with the [Toolbar](#) style's pop-up menu property option.

By default, the OS X Gradient style has a light gray background with a label in the center. The appearance of the button can be different when the cursor hovers over it depending on the OS:

- *Windows* - the button is identical to the Bevel style. When it uses the "With Pop-

up Menu" property, a triangle is displayed on the right side of the button.



- *macOS* - the button is displayed as a two-tone system button. When it uses the "With Pop-up Menu" property, a triangle is displayed to the right and at the bottom of the button.

### JSON Example:

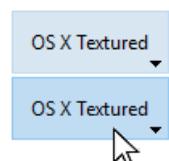
```
"myButton": {  
    "type": "button",  
    "style": "gradientBevel",  
    "text": "OK",  
    "popupPlacement": "linked",  
    "action": "Cancel",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

### OS X Textured

The OS X Textured button style is nearly identical to the [Bevel](#) style but with a smaller size (maximum size is the size of a standard macOS system button). As with the Bevel style, the OS X Textured style combines the appearance of the [Regular](#) style with the [Toolbar](#) style's pop-up menu property option.

By default, the OS X Textured style appears as:

- *Windows* - a standard system button with a light gray background with a label in the center. It has the special feature of being transparent in Vista.



- *macOS* - a standard system button displaying a color change from light to dark gray. Its height is predefined: it is not possible to enlarge or reduce it.

### JSON Example:

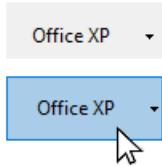
```
"myButton": {  
    "type": "button",  
    "style": "texturedBevel",  
    "text": "OK",  
    "popupPlacement": "separated",  
    "action": "Cancel",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

### Office XP

The Office XP button style combines the appearance of the [Regular](#) style with the [Toolbar](#) style's transparency and pop-up menu property option.

The colors (highlight and background) of a button with the Office XP style are based on the system colors. The appearance of the button can be different when the cursor hovers over it depending on the OS:

- *Windows* - its background only appears when the mouse rolls over it.



- *macOS* - its background is always displayed.

### JSON Example:

```
"myButton": {  
    "type": "button",  
    "style": "office",  
    "text": "OK",  
    "popupPlacement": "none",  
    "action": "Cancel",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

### Help

The Help button style can be used to display a standard system help button. By default, the Help style is displayed as a question mark within a circle.



### JSON Example:

```
"myButton": {  
    "type": "button",  
    "style": "help",  
    "text": "OK",  
    "dropping": "custom",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

The Help style does not support [Number of States](#), [Picture pathname](#), and [Title/Picture Position](#) basic properties.

### Circle

The Circle button style appears as a round system button. This button style is designed for macOS.



On Windows, it is identical to the "None" style (the circle in the background is not taken into account).

### JSON Example:

```
"myButton": {  
    "type": "button",  
    "style": "circular",  
    "text": "OK",  
    "dropping": "custom",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

## Custom

The Custom button style accepts a personalized background picture and allows managing additional parameters such as icon and margin offset.



### JSON Example:

```
"myButton": {  
    "type": "button",  
    "style": "custom",  
    "text": "",  
    "customBackgroundPicture": "/RESOURCES/bkgnd.png",  
    "icon": "/RESOURCES/custom.png",  
    "textPlacement": "center",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

## Supported Properties

All buttons share the same set of basic properties:

[Bold](#) - [Border Line Style](#) - [Bottom](#) - [Button Style](#) - [Class](#) - [Droppable](#) - [Focusable](#) - [Font](#) - [Font Color](#) - [Font Size](#) - [Height](#) - [Help Tip](#) - [Horizontal Alignment](#) - [Horizontal Sizing](#) - [Italic](#) - [Image hugs title](#)(1) - [Left](#) - [Not rendered](#) - [Number of States](#)(1) - [Object Name](#) - [Picture pathname](#)(1) - [Right](#) - [Shortcut](#) - [Standard action](#) - [Title](#) - [Title/Picture](#) [Position](#)(1) - [Top](#) - [Type](#) - [Underline](#) - [Variable or Expression](#) - [Vertical Sizing](#) - [Visibility](#) - [Width](#) - [With pop-up menu](#)(2)

- (1) Not supported by the [Help](#) style.
- (2) Not supported by the [Help](#), [Flat](#) and [Regular](#) styles.

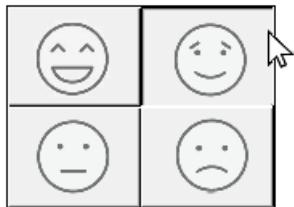
Additional specific properties are available, depending on the [button style](#):

- Custom: [Background pathname](#) - [Horizontal Margin](#) - [Icon Offset](#) - [Vertical Margin](#)
- Flat, Regular: [Default Button](#)



## Button Grid

A button grid is a transparent object that is placed on top of a graphic. The graphic should depict a row-by-column array. When one of the graphics is clicked on, it will have a sunken or pressed appearance:



You can use a button grid object to determine where the user clicks on the graphic. The object method would use the `On Clicked` event and take appropriate action depending on the location of the click.

## Creating button grids

To create the button grid, add a background graphic to the form and place a button grid on top of it. Specify the number of [rows](#) and [columns](#).

In 4D, a button grid is used as a color palette:

## Using button grids

The buttons on the grid are numbered from top left to bottom right. In the above example, the grid is 16 columns across by 16 rows down. The button in the top-left position returns 1 when clicked. If the red button at the far right of the second row is selected, the button grid returns 32. If no element is selected, the value is 0

### Goto page

You can assign the `gotoPage` [standard action](#) to a button grid. When this action is selected, 4D will automatically display the page of the form that corresponds to the number of the button that is selected in the button grid. For example, if the user selects the tenth button of the grid, 4D will display the tenth page of the current form (if it exists).

## Supported Properties

[Border Line Style](#) - [Bottom](#) - [Class](#) - [Columns](#) - [Height](#) - [Help Tip](#) - [Horizontal Sizing](#) - [Left](#) - [Object Name](#) - [Right](#) - [Rows](#) - [Standard action](#) - [Top](#) - [Type](#) - [Variable or Expression](#) - [Vertical Sizing](#) - [Width](#) - [Visibility](#)

## Check Box

A check box is a type of button used to enter or display binary (true-false) data. Basically, it is either checked or unchecked, but a [third state](#) can be defined.



Check boxes are controlled by methods or [standard actions](#). The method associated with it executes when the check box is selected. Like all buttons, a check box variable is set to 0 when the form is first opened.

A check box displays text next to a small square. This text is set in the [Title](#) property of the check box. You can enter a title in the form of an XLIFF reference in this area (see [Appendix B: XLIFF architecture](#)).

## Using check boxes

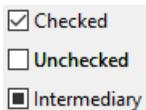
A check box can be associated to a [variable or expression](#) of type integer or boolean.

- **integer:** if the box is checked, the variable has the value 1. When not checked, it has the value 0. If check box is in third state (see below), it has the value 2.
- **boolean:** if the box is checked, the variable has the value `True`. When not checked, it has the value `False`.

Any or all check boxes in a form can be checked or unchecked. Multiple check boxes allow the user to select multiple options.

## Three-States check box

Check box objects with [Regular](#) and [Flat button style](#) accept a third state. This third state is an intermediate status, which is generally used for display purposes. For example, it allows indicating that a property is present in a selection of objects, but not in each object of the selection.



To enable this third state, you must select the [Three-States](#) property.

This property is only available for regular and flat check boxes associated with numeric [variables or expressions](#) — check boxes for Boolean expressions cannot use the [Three-States](#) property (a Boolean expression cannot be in an intermediary state).

The variable associated with the check box returns the value 2 when the check box is in the third state.

In entry mode, the Three-States check boxes display each state sequentially, in the following order: unchecked / checked / intermediary / unchecked, etc. The intermediary state is generally not very useful in entry mode; in the code, simply force the value of the variable to 0 when it takes the value of 2 in order to pass directly from the checked state to the unchecked state.

## Using a standard action

You can assign a [standard action](#) to a check box to handle attributes of text areas. For example, if you assign the `fontBold` standard action, at runtime the check box will manage the "bold" attribute of the selected text in the current area.

Only actions that can represent a true/false status ("checkable" actions) are supported by this object:

| <b>Supported actions</b>            | <b>Usage condition (if any)</b> |
|-------------------------------------|---------------------------------|
| avoidPageBreakInsideEnabled         | 4D Write Pro areas only         |
| fontItalic                          |                                 |
| fontBold                            |                                 |
| fontLinethrough                     |                                 |
| fontSubscript                       | 4D Write Pro areas only         |
| fontSuperscript                     | 4D Write Pro areas only         |
| fontUnderline                       |                                 |
| font/showDialog                     | Mac only                        |
| htmlWYSIWIGEnabled                  | 4D Write Pro areas only         |
| section/differentFirstPage          | 4D Write Pro areas only         |
| section/differentLeftRightPages     | 4D Write Pro areas only         |
| spell/autoCorrectionEnabled         |                                 |
| spell/autoDashSubstitutionsEnabled  | Mac only                        |
| spell/autoLanguageEnabled           | Mac only                        |
| spell/autoQuoteSubstitutionsEnabled | Mac only                        |
| spell/autoSubstitutionsEnabled      |                                 |
| spell/enabled                       |                                 |
| spell/grammarEnabled                | Mac only                        |
| spell/showDialog                    | Mac only                        |
| spell/visibleSubstitutions          |                                 |
| visibleBackground                   | 4D Write Pro areas only         |
| visibleFooters                      | 4D Write Pro areas only         |
| visibleHeaders                      | 4D Write Pro areas only         |
| visibleHiddenChars                  | 4D Write Pro areas only         |
| visibleHorizontalRuler              | 4D Write Pro areas only         |
| visiblePageFrames                   | 4D Write Pro areas only         |
| visibleReferences                   |                                 |
| widowAndOrphanControlEnabled        | 4D Write Pro areas only         |

For detailed information on these actions, please refer to the [Standard actions](#) section.

## Check box button styles

Check boxes use [button styles](#) to control a check box's general appearance as well as its available properties. It is possible to apply different predefined styles to check boxes. A great number of variations can be obtained by combining these properties / behaviors.

With the exception of the [available properties](#), many check box objects are *structurally* identical. The difference is in the processing of their associated variables.

4D provides check boxes in the following predefined button styles:

### Regular

The Regular check box button style is a standard system check box (*i.e.*, a rectangle with a descriptive title):

Checkbox Checkbox

### JSON Example:

```
"myCheckBox": {  
    "type": "checkbox",  
    "style": "regular",  
    "text": "Cancel",  
    "action": "Cancel",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
    "dataSourceTypeHint": "boolean"  
}
```

### Flat

The Flat check box button style is a minimalist appearance. The Flat style's graphic nature is particularly useful for forms that will be printed.

 Checkbox Checkbox

### JSON Example:

```
"myCheckBox": {  
    "type": "checkbox",  
    "style": "flat",  
    "text": "Cancel",  
    "action": "cancel",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

### Toolbar Button

The Toolbar Button check box button style is primarily intended for integration in a toolbar.

The Toolbar Button check box button style has a transparent background with a title. It is usually associated with a [4-state picture](#).

Example with states unchecked / checked / highlighted:



### JSON Example:

```

"myCheckBox": {
    "type": "checkbox",
    "style": "toolbar",
    "text": "Checkbox",
    "icon": "/RESOURCES/File.png",
    "iconFrames": 4
    "left": 60,
    "top": 160,
    "width": 100,
    "height": 20
}

```

## Bevel

The Bevel check box button style combines the appearance of the [Regular](#) button style (*i.e.*, a rectangle with a descriptive title) with the [Toolbar Button](#) button style's behavior.

The Bevel button style has a light gray background with a title. It is usually associated with a [4-state picture](#).

Example with states unchecked / checked / highlighted:



## JSON Example:

```

"myCheckBox": {
    "type": "checkbox",
    "style": "bevel",
    "text": "Checkbox",
    "icon": "/RESOURCES/File.png",
    "iconFrames": 4
    "left": 60,
    "top": 160,
    "width": 100,
    "height": 20
}

```

## Rounded Bevel

The Rounded Bevel check box button style is nearly identical to the [Bevel](#) button style except, depending on the OS, the corners of the button may be rounded. As with the Bevel button style, the Rounded Bevel button style combines the appearance of the [Regular](#) button style with the [Toolbar Button](#) button style's behavior.

The Rounded Bevel button style has a light gray background with a title. It is usually associated with a [4-state picture](#).

Example on macOS:

On Windows, the Rounded Bevel button style is identical to the [Bevel](#) button style.

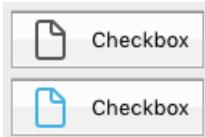
### JSON Example:

```
"myCheckBox": {  
    "type": "checkbox",  
    "style": "roundedBevel",  
    "text": "Checkbox",  
    "icon": "/RESOURCES/File.png",  
    "iconFrames": 4  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

### OS X Gradient

The OS X Gradient check box button style is nearly identical to the [Bevel](#) button style. As with the Bevel button style, the OS X Gradient button style combines the appearance of the [Regular](#) button style with the [Toolbar Button](#) button style's behavior.

The OS X Gradient button style has a light gray background with a title and may be displayed as a two-tone system button on macOS. It is usually associated with a [4-state picture](#).



On Windows, this check box button style is identical to the [Bevel](#) button style.

### JSON Example:

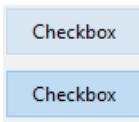
```
"myCheckBox": {  
    "type": "checkbox",  
    "style": "gradientBevel",  
    "text": "Checkbox",  
    "icon": "/RESOURCES/File.png",  
    "iconFrames": 4  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

### OS X Textured

The OS X Textured button style is similar to the [Bevel](#) button style but with a smaller size (maximum size is the size of a standard macOS system button). As with the Bevel button style, the OS X Textured button style combines the appearance of the [Regular](#) button style with the [Toolbar Button](#) button style's behavior.

By default, the OS X Textured button style appears as:

- *Windows* - a standard system button with a light blue background with a title in the center.



- *macOS* - a standard system button. Its height is predefined: it is not possible to enlarge or reduce it.



### JSON Example:

```
"myCheckBox": {  
    "type": "checkbox",  
    "style": "texturedBevel",  
    "text": "Checkbox",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

### Office XP

The Office XP button style combines the appearance of the [Regular](#) button style with the [Toolbar Button](#) button style's behavior.

The colors (highlight and background) of a check box with the Office XP button style are based on the system colors. The appearance of the check box can be different when the cursor hovers over it, depending on the OS:

- *Windows* - its background only appears when the mouse rolls over it. Example with states unchecked / checked / highlighted:



- *macOS* - its background is always displayed. Example with states unchecked / checked:

### JSON Example:

```

"myCheckBox": {
    "type": "checkbox",
    "style": "office",
    "text": "Checkbox",
    "action": "fontBold",
    "icon": "/RESOURCES/File.png",
    "iconFrames": 4
    "left": 60,
    "top": 160,
    "width": 100,
    "height": 20
}

```

## Collapse/Expand

This check box button style can be used to add a standard collapse/expand icon. These icons are used natively in hierarchical lists.

- *Windows* - the icon looks like a [+] or a [-]



- *macOS* - it looks like a triangle pointing right or down.



### INFO

The Collapse/Expand style is named "disclosure" in the [button style JSON Grammar](#).

## JSON Example:

```

"myCheckBox": {
    "type": "checkbox",
    "style": "disclosure",
    "method": "mCollapse",
    "left": 60,
    "top": 160,
    "width": 100,
    "height": 20
}

```

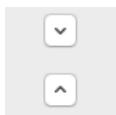
## Disclosure

In macOS and Windows, a check box with the "Disclosure" button style appears as a standard disclosure button, usually used to show/hide additional information. When used as a radio button, the button symbol points downwards with value 0 and upwards with value 1.

- *Windows*



- *macOS*



## ⓘ INFO

The Disclosure style is named "roundedDisclosure" in the [button style JSON Grammar](#).

### JSON Example:

```
"myCheckBox": {  
    "type": "checkbox",  
    "style": "roundedDisclosure",  
    "method": "m_disclose",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

## Custom

The Custom button style accepts a personalized background picture and allows managing specific properties:

- [Background pathname](#)
- [Icon Offset](#)
- [Horizontal Margin](#) and [Vertical Margin](#)

It is usually associated with a [4-state picture](#), that can be used in conjunction with a [4-state background picture](#).

### JSON Example:

```
"myCheckbox": {  
    "type": "checkbox",  
    "style": "custom",  
    "text": "OK",  
    "icon": "/RESOURCES/smiley.jpg",  
    "iconFrame": 4,  
    "customBackgroundPicture": "/RESOURCES/paper.jpg",  
    "iconOffset": 5, //custom icon offset when clicked  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20,  
    "customBorderX": 20,  
    "customBorderY": 5  
}
```

## Supported Properties

All check boxes share the same set of basic properties:

[Bold](#) - [Bottom](#) - [Button Style](#) - [Class](#) - [Enterable](#) - [Expression Type](#) - [Focusable](#) - [Font](#) - [Font Color](#) - [Font Size](#) - [Height](#) - [Help Tip](#) - [Horizontal Alignment\(1\)](#) - [Horizontal Sizing](#) - [Image hugs title\(2\)](#) - [Italic](#) - [Left](#) - [Number of States\(2\)](#) - [Object Name](#) - [Picture pathname\(2\)](#) - [Right](#) - [Save value](#) - [Shortcut](#) - [Standard action](#) - [Title](#) - [Title/Picture Position\(2\)](#) - [Top](#) - [Type](#) - [Underline](#) - [Variable or Expression](#) - [Vertical Sizing](#) - [Visibility](#) - [Width](#)

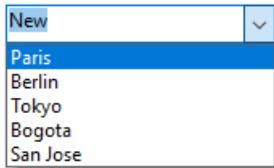
- (1) Not supported by the [Regular](#) and [Flat](#) styles.
- (2) Not supported by the [Regular](#), [Flat](#), [Disclosure](#) and [Collapse/Expand](#) styles.

Additional specific properties are available, depending on the [button style](#):

- Custom: [Background pathname](#) - [Horizontal Margin](#) - [Icon Offset](#) - [Vertical Margin](#)
- Flat, Regular: [Three-States](#)

## Combo Box

A combo box is similar to a [drop-down list](#), except that it accepts text entered from the keyboard and has additional options.



Fundamentally, you treat a combo box as an enterable area that uses its object, array or a choice list as the set of default values.

### Handling combo boxes

Use the [On Data Change](#) event to manage entries into the enterable area, as you would for any input form object.

You initialize a combo box in exactly the same way as a [drop-down list](#): using an object, an array, or a choice list.

#### Using an object

This feature is only available in 4D projects.

An [object](#) encapsulating a [collection](#) can be used as the data source of a combo box. The object must contain the following properties:

| Property     | Type                  | Description                                                                                                                                                                                                                                                |
|--------------|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| values       | Collection            | Mandatory - Collection of scalar values. All values must be of the same type. Supported types: <ul style="list-style-type: none"><li>• strings</li><li>• numbers</li><li>• dates</li><li>• times</li></ul> If empty or not defined, the combo box is empty |
| currentValue | same as<br>Collection | Text entered by the user                                                                                                                                                                                                                                   |

If the object contains other properties, they are ignored.

When the user enters text into the combo box, the `currentValue` property of the object gets the entered text.

#### Using an array

Please refer to **Using an array** in the [drop-down list page](#) for information about how to initialize the array.

When the user enters text into the combo box, the 0th element of the array gets the entered text.

#### Using a choice list

If you want to use a combo box to manage the values of an input area (listed field or variable), 4D lets you reference the field or variable directly as the form object's data

source. This makes it easier to manage listed fields/variables.

If you use a hierarchical list, only the first level is displayed and can be selected.

To associate a combo box with a field or variable, you can just enter the name of the field or variable directly in the [Variable or Expression](#) of the form object in the Property List.

When the form is executed, 4D automatically manages the combo box during input or display: when a user chooses a value, it is saved in the field; this field value is shown in the combo box when the form is displayed:

Please refer to **Using a choice** in the [drop-down list page](#) for more information.

## Options

Combo box type objects accept two specific options:

- [Automatic insertion](#): enables automatically adding a value to the data source when a user enters a value that is not found in the list associated with the combo box.
- [Excluded List](#) (list of excluded values): allows setting a list whose values cannot be entered in the combo box. If an excluded value is entered, it is not accepted and an error message is displayed.

Associating a [list of required values](#) is not available for combo boxes. In an interface, if an object must propose a finite list of required values, then you must use a [drop-down list](#) object.

## Supported Properties

[Alpha Format](#) - [Bold](#) - [Bottom](#) - [Choice List](#) - [Class](#) - [Date Format](#) - [Expression Type](#) - [Font](#) - [Font Color](#) - [Font Size](#) - [Height](#) - [Help Tip](#) - [Horizontal Sizing](#) - [Italic](#) - [Left](#) - [Object Name](#) - [Right](#) - [Time Format](#) - [Top](#) - [Type](#) - [Underline](#) - [Variable or Expression](#) - [Vertical Sizing](#) - [Visibility](#) - [Width](#)

## Drop-down List

Drop-down lists are form objects that allow the user to select from a list. You manage the items displayed in the drop-down list using an object, an array, a choice list, or a standard action.

On macOS, drop-down lists are also sometimes called "pop-up menu". Both names refer to the same objects. As the following example shows, the appearance of these objects can differ slightly according to the platform:

## Drop-down list types

You can create different types of drop-down lists with different features. To define a type, select the appropriate **Expression Type** and **Data Type** values in the Property list, or use their JSON equivalent.

| Type                           | Features                                           | Expression Type                                     | Data Type                                                                                                                               | JSON definition                                                                                                                                                  |
|--------------------------------|----------------------------------------------------|-----------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Object                         | Built upon a collection                            | Object                                              | Numeric, Text, Date, or Time                                                                                                            | <code>dataSourceTypeHint: object + numberFormat: &lt;format&gt; or textFormat: &lt;format&gt; or dateFormat: &lt;format&gt; or timeFormat: &lt;format&gt;</code> |
| Array                          | Built upon an array                                | Array                                               | Numeric, Text, Date, or Time                                                                                                            | <code>dataSourceTypeHint: arrayNumber or arrayText or arrayDate or arrayTime</code>                                                                              |
| Choice list saved as value     | Built upon a choice list (standard)                | List                                                | Selected item value                                                                                                                     | <code>dataSourceTypeHint: text + saveAs: value</code>                                                                                                            |
| Choice list saved as reference | Built upon a choice list. Item position is saved   | List                                                | Selected item reference                                                                                                                 | <code>dataSourceTypeHint: integer + saveAs: reference</code>                                                                                                     |
| Hierarchical choice list       | Can display hierarchical contents                  | List                                                | List reference                                                                                                                          | <code>dataSourceTypeHint: integer</code>                                                                                                                         |
| Standard action                | Automatically built by the <code>any</code> action | <code>any</code> except <code>List</code> reference | <code>any</code> definition + <code>action: &lt;action&gt;</code> (+ <code>focusable: false</code> for actions applying to other areas) |                                                                                                                                                                  |

## Handling drop-down lists

### Using an object

This feature is only available in 4D projects.

An [object](#) encapsulating a [collection](#) can be used as the data source of a drop-down list. The object must contain the following properties:

| Property                  | Type                            | Description                                                                                                                                                                                                                                                     |
|---------------------------|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>values</code>       | Collection                      | Mandatory - Collection of scalar values. All values must be of the same type. Supported types: <ul style="list-style-type: none"><li>• strings</li><li>• numbers</li><li>• dates</li><li>• times</li></ul> If empty or not defined, the drop-down list is empty |
| <code>index</code>        | number                          | Index of the currently selected item (value between 0 and <code>collection.length-1</code> ). If you set -1, <code>currentValue</code> is displayed as a placeholder string                                                                                     |
| <code>currentValue</code> | same as <code>Collection</code> | Currently selected item (used as placeholder value if set by code)                                                                                                                                                                                              |

If the object contains other properties, they are ignored.

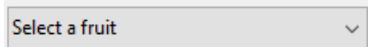
To initialize the object associated to the drop-down list, you can:

- Enter a list of default values in the object properties by selecting `\<Static List>` in the [Data Source](#) theme of the Property List. The default values are loaded into an object automatically.
- Execute code that creates the object and its properties. For example, if "myList" is the [variable](#) associated to the drop-down list, you can write in the [On Load](#) form event:

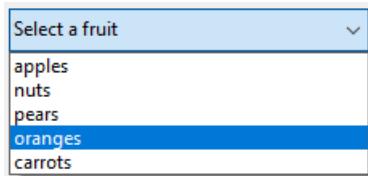
```
// Form.myDrop is the datasource of the form object
```

```
Form.myDrop:=New object
Form.myDrop.values:=New collection("apples"; "nuts"; "pears";
"oranges"; "carrots")
Form.myDrop.index:=-1 //currentValue is a placeholder
Form.myDrop.currentValue:="Select a fruit"
```

The drop-down list is displayed with the placeholder string:



After the user selects a value:



```
Form.myDrop.values // ["apples", "nuts", "pears", "oranges", "carrots"]
Form.myDrop.currentValue // "oranges"
Form.myDrop.index // 3
```

## Using an array

An [array](#) is a list of values in memory that is referenced by the name of the array. A drop-down list can display an array as a list of values when you click on it.

To initialize the array associated to the drop-down list, you can:

- Enter a list of default values in the object properties by selecting `\<Static List>` in the [Data Source](#) theme of the Property List. The default values are

loaded into an array automatically. You can refer to the array using the name of the variable associated with the object.

- Before the object is displayed, execute code that assigns values to the array elements. For example:

```
ARRAY TEXT(aCities;6)
aCities{1} := "Philadelphia"
aCities{2} := "Pittsburg"
aCities{3} := "Grand Blanc"
aCities{4} := "Bad Axe"
aCities{5} := "Frostbite Falls"
aCities{6} := "Green Bay"
```

In this case, the name of the [variable](#) associated with the object in the form must be `aCities`. This code could be placed in the form method and be executed when the [On Load](#) form event runs.

- Before the object is displayed, load the values of a list into the array using the [LIST TO ARRAY](#) command. For example:

```
LIST TO ARRAY("Cities";aCities)
```

In this case also, the name of the [variable](#) associated with the object in the form must be `aCities`. This code would be run in place of the assignment statements shown above.

If you need to save the user's choice into a field, you would use an assignment statement that runs after the record is accepted. The code might look like this:

```
Case of
  :(Form event=On Load)
    LIST TO ARRAY("Cities";aCities)
    If(Record number([People])<0) `new record
      aCities:=3 `display a default value
    Else `existing record, display stored value
      aCities:=Find in array(aCities;City)
    End if
  :(Form event=On Clicked) `user modified selection
    City:=aCities{aCities} `field gets new value
  :(Form event=On Validate)
    City:=aCities{aCities}
  :(Form event=On Unload)
    CLEAR VARIABLE(aCities)
End case
```

You must select each event that you test for in your Case statement. Arrays always contain a finite number of items. The list of items is dynamic and can be changed by a method. Items in an array can be modified, sorted, and added to.

## Using a choice list

If you want to use a drop-down list to manage the values of an input area (listed field or variable), 4D lets you reference the field or variable directly as the drop-down list's [data source](#). This makes it easier to manage listed fields/variables.

For example, in the case of a "Color" field that can only contain the values "White", "Blue", "Green" or "Red", it is possible to create a list containing these values and associate it with a drop-down list that references the 4D "Color" field. 4D then automatically takes care of managing the input and display of the current value in the form.

If you use a hierarchical list, only the first level is displayed and can be selected. If you want to display hierarchical contents, you need to use a [hierarchical choice list](#).

To associate a drop-down list with a field or variable, enter the name of the field or variable directly as the [Variable or Expression](#) field of the drop-down list in the Property List.

It is not possible to use this feature with an object or an array drop-down list. If you enter a field name in the "Variable or Expression" area, then you must use a choice list.

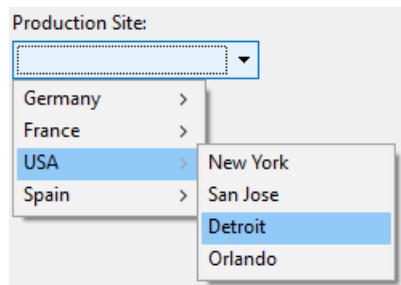
When the form is executed, 4D automatically manages the drop-down list during input or display: when a user chooses a value, it is saved in the field; this field value is shown in the drop-down list when the form is displayed:

## Selected item value or Selected item reference

When you have associated a drop-down list with a choice list and with a field or a variable, you can set the [Data Type](#) property to **Selected item value** or **Selected item reference**. This option lets you optimize the size of the data saved.

## Using a hierarchical choice list

A hierarchical drop-down list has a sublist associated with each item in the list. Here is an example of a hierarchical drop-down list:



In forms, hierarchical drop-down lists are limited to two levels.

You can assign the hierarchical choice list to the drop-down list object using the [Choice List](#) field of the Property List.

You manage hierarchical drop-down lists using the **Hierarchical Lists** commands of the 4D Language. All commands that support the `(*; "name")` syntax can be used with hierarchical drop-down lists, e.g. [List item parent](#).

## Using a standard action

You can build automatically a drop-down list using a [standard action](#). This feature is supported in the following contexts:

- Use of the `gotoPage` standard action. In this case, 4D will automatically display the [page of the form](#) that corresponds to the number of the item that is selected. For example, if the user selects the 3rd item, 4D will display the third page of the current form (if it exists). At runtime, by default the drop-down list displays the page numbers (1, 2...).
- Use of a standard action that displays a sublist of items, for example

`backgroundColor`. This feature requires that:

- a styled text area ([4D Write Pro area](#) or [input](#) with [multistyle](#) property) is present in the form as the standard action target.
- the [focusable](#) property is not set to the drop-down list. At runtime the drop-down list will display an automatic list of values, e.g. background colors. You can override this automatic list by assigning in addition a choice list in which each item has been assigned a custom standard action.

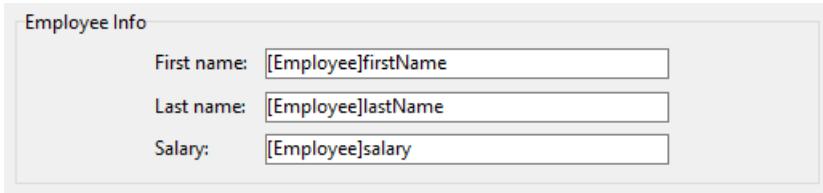
This feature cannot be used with a hierarchical drop-down list.

## Supported Properties

[Alpha Format](#) - [Bold](#) - [Bottom](#) - [Button Style](#) - [Choice List](#) - [Class](#) - [Data Type \(expression type\)](#) - [Data Type \(list\)](#) - [Date Format](#) - [Expression Type](#) - [Focusable](#) - [Font](#) - [Font Color](#) - [Font Size](#) - [Height](#) - [Help Tip](#) - [Horizontal Sizing](#) - [Italic](#) - [Left](#) - [Not rendered](#) - [Object Name](#) - [Right](#) - [Standard action](#) - [Save value](#) - [Time Format](#) - [Top](#) - [Type](#) - [Underline](#) - [Variable or Expression](#) - [Vertical Sizing](#) - [Visibility](#) - [Width](#)

## Group Box

A group box is a static object that allows you to visually assemble multiple form objects:



The name of a group box is static text; you can use a “localizable” reference as with any 4D label (see [Using references in static text](#) and *XLIFF Architecture* section in 4D Design Reference.

### JSON Example:

```
"myGroup": {  
    "type": "groupBox",  
    "title": "Employee Info"  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

### Supported Properties

[Bottom](#) - [CSS Class](#) - [Font](#) - [Font Color](#) - [Font Size](#) - [Height](#) - [Horizontal Alignment](#) - [Horizontal Sizing](#) - [Italic](#) - [Left](#) - [Object Name](#) - [Right](#) - [Title](#) - [Top](#) - [Type](#) - [Underline](#) - [Vertical Sizing](#) - [Visibility](#) - [Width](#)

# Input

Inputs allow you to add enterable or non-enterable expressions such as database [fields](#) and [variables](#) to your forms. Inputs can handle character-based data (text, dates, numbers...) or pictures:

Inputs can contain [assignable or non-assignable expressions](#).

In addition, inputs can be [enterable or non-enterable](#). An enterable input accepts data. You can set data entry controls for the object. A non-enterable input can only display values but cannot be edited by the user.

You can manage the data with object or form [methods](#).

## JSON Example:

```
"myText": {  
    "type": "input",      //define the type of object  
    "spellcheck": true, //enable spelling verification  
    "left": 60,          //left position on the form  
    "top": 160,          //top position on the form  
    "width": 100,         //width of the object  
    "height": 20          //height of the object  
}
```

## Supported Properties

- History

[Allow font/color picker](#) - [Alpha Format](#) - [Auto Spellcheck](#) - [Bold](#) - [Text when False/Text when True](#) - [Border Line Style](#) - [Bottom](#) - [Choice List](#) - [Class](#) - [Context Menu](#) - [Corner radius](#) - [Date Format](#) - [Default value](#) - [Draggable](#) - [Droppable](#) - [Enterable](#) - [Entry Filter](#) - [Excluded List](#) - [Expression type](#) - [Fill Color](#) - [Font](#) - [Font Color](#) - [Font Size](#) - [Height](#) - [Hide focus rectangle](#) - [Horizontal Alignment](#) - [Horizontal Scroll Bar](#) - [Horizontal Sizing](#) - [Italic](#) - [Left](#) - [Multiline](#) - [Multi-style](#) - [Number Format](#) - [Object Name](#) - [Orientation](#) - [Picture Format](#) - [Placeholder](#) - [Print Frame](#) - [Required List](#) - [Right](#) - [Selection always visible](#) - [Store with default style tags](#) - [Text when False/Text when True](#) - [Time Format](#) - [Top](#) - [Type](#) - [Underline](#) - [Variable or Expression](#) - [Vertical Scroll Bar](#) - [Vertical Sizing](#) - [Visibility](#) - [Width](#) - [Wordwrap](#)

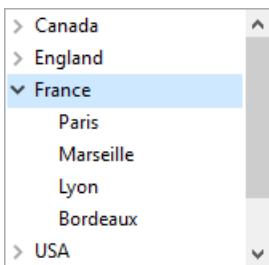
## Input alternatives

You can also represent field and variable expressions in your forms using alternative objects, more particularly:

- You can display and enter data from database fields directly in columns of [selection type List boxes](#).
- You can represent a list field or variable directly in a form using [Pop-up Menus/Drop-down Lists](#) and [Combo Boxes](#) objects.
- You can represent a boolean expression as a [check box](#) or as a [radio button](#) object.

## Hierarchical List

Hierarchical lists are form objects that can be used to display data as lists with one or more levels that can be expanded or collapsed.



Where appropriate, the expand/collapse icon is automatically displayed to the left of the item. Hierarchical lists support an unlimited number of sublevels.

## Hierarchical list data source

The contents of a hierarchical list form object can be initialized in one of the following ways:

- Associate an existing [choice list](#) to the object. The choice list must have been defined in the List editor in Design mode.
- Directly assign a hierarchical list reference to the [variable or expression](#) associated with the form object.

In both cases, you manage a hierarchical list at runtime through its *ListRef* reference, using the [Hierarchical list](#) commands in the 4D language.

## ListRef and object name

A hierarchical list is both a **language object** existing in memory and a **form object**.

The **language object** is referenced by an unique internal ID of the Longint type, designated by *ListRef* in the 4D Language Reference. This ID is returned by the commands that can be used to create lists: `New list`, `Copy list`, `Load list`, `BLOB to list`. There is only one instance of the language object in memory and any modification carried out on this object is immediately carried over to all the places where it is used.

The **form object** is not necessarily unique: there may be several representations of the same hierarchical list in the same form or in different ones. As with other form objects, you specify the object in the language using the syntax (\*;"*ObjectName*", etc.).

You connect the hierarchical list "language object" with the hierarchical list "form object" by the intermediary of the variable containing the *ListRef* value. For example, if you have associated the `mylist` [variable](#) to the form object, you can write:

```
mylist:=New list
```

Each representation of the list has its own specific characteristics and shares common characteristics with all the other representations. The following characteristics are specific to each representation of the list:

- The selection,
- The expanded/collapsed state of its items,

- The position of the scrolling cursor.

The other characteristics (font, font size, style, entry control, color, list contents, icons, etc.) are common to all the representations and cannot be modified separately. Consequently, when you use commands based on the expanded/collapsed configuration or the current item, for example `Count list items` (when the final `*` parameter is not passed), it is important to be able to specify the representation to be used without any ambiguity.

You must use the `ListRef` ID with language commands when you want to specify the hierarchical list found in memory. On the other hand, if you want to specify the representation of a hierarchical list object at the form level, you must use the object name (string type) in the command, via the standard syntax (`*;"ListName"`, etc.).

In the case of commands that set properties, the syntax based on the object name does not mean that only the form object specified will be modified by the command, but rather that the action of the command will be based on the state of this object. The common characteristics of hierarchical lists are always modified in all of their representations. For example, if you execute:

```
SET LIST ITEM FONT(*;"mylist1";*;thefont)
```

... you are indicating that you want to modify the font of the hierarchical list item associated with the *mylist1* form object. The command will take the current item of the *mylist1* object into account to specify the item to modify, but this modification will be carried over to all the representations of the list in all of the processes.

## **Support of @**

As with other object property management commands, it is possible to use the "@" character in the `ListName` parameter. As a rule, this syntax is used to designate a set of objects in the form. However, in the context of hierarchical list commands, this does not apply in every case. This syntax will have two different effects depending on the type of command:

- For commands that set properties, this syntax designates all the objects whose name corresponds (standard behavior). For example, the parameter "LH@" designates all objects of the hierarchical list type whose name begins with "LH."
  - `DELETE FROM LIST`
  - `INSERT IN LIST`
  - `SELECT LIST ITEMS BY POSITION`
  - `SET LIST ITEM`
  - `SET LIST ITEM FONT`
  - `SET LIST ITEM ICON`
  - `SET LIST ITEM PARAMETER`
  - `SET LIST ITEM PROPERTIES`
- For commands retrieving properties, this syntax designates the first object whose name corresponds:
  - `Count list items`
  - `Find in list`
  - `GET LIST ITEM`
  - `Get list item font`
  - `GET LIST ITEM ICON`

- GET LIST ITEM PARAMETER
- GET LIST ITEM PROPERTIES
- List item parent
- List item position
- Selected list items

## Generic commands to use with hierarchical lists

It is possible to modify the appearance of a hierarchical list form objects using several generic 4D commands. You can pass to these commands either the object name of the hierarchical list (using the \* parameter), or its variable name (containing the ListRef value):

- OBJECT SET FONT
- OBJECT SET FONT STYLE
- OBJECT SET FONT SIZE
- OBJECT SET COLOR
- OBJECT SET FILTER
- OBJECT SET ENTERABLE
- OBJECT SET SCROLLBAR
- OBJECT SET SCROLL POSITION
- OBJECT SET RGB COLORS

Reminder: Except OBJECT SET SCROLL POSITION, these commands modify all the representations of the same list, even if you only specify a list via its object name.

## Priority of property commands

Certain properties of hierarchical lists (for example, the **Enterable** attribute or the color) can be set in different ways: in the form properties, via a command of the “Object Properties” theme or via a command of the “Hierarchical Lists” theme. When all three of these means are used to set list properties, the following order of priority is applied:

1. Commands of the “Hierarchical Lists” theme
2. Generic object property commands
3. Form property

This principle is applied regardless of the order in which the commands are called. If an item property is modified individually via a hierarchical list command, the equivalent object property command will have no effect on this item even if it is called subsequently. For example, if the color of an item is modified via the SET LIST ITEM PROPERTIES command, the OBJECT SET COLOR command will have no effect on this item.

## Management of items by position or by reference

You can usually work in two ways with the contents of hierarchical lists: by position or by reference.

- When you work by position, 4D bases itself on the position in relation to the items of the list displayed on screen in order to identify them. The result will differ according to whether or not certain hierarchical items are expanded or collapsed. Note that in the case of multiple representations, each form object has its own configuration of expanded/collapsed items.

- When you work by reference, 4D bases itself on the *itemRef* ID number of the list items. Each item can thus be specified individually, regardless of its position or its display in the hierarchical list.

## Using item reference numbers (*itemRef*)

Each item of a hierarchical list has a reference number (*itemRef*) of the Longint type. This value is only intended for your own use: 4D simply maintains it.

**Warning:** You can use any type of Longint value as a reference number, except for 0. In fact, for most of the commands in this theme, the value 0 is used to specify the last item added to the list.

Here are a few tips for using reference numbers:

1. You do not need to identify each item with a unique number (beginner level).
  - First example: you build a system of tabs by programming, for example, an address book. Since the system returns the number of the tab selected, you will probably not need more information than this. In this case, do not worry about item reference numbers: pass any value (except 0) in the *itemRef* parameter. Note that for an address book system, you can predefined a list A, B, ..., Z in Design mode. You can also create it by programming in order to eliminate any letters for which there are no records.
  - Second example: while working with a database, you progressively build a list of keywords. You can save this list at the end of each session by using the `SAVE LIST` or `LIST TO BLOB` commands and reload it at the beginning of each new session using the `Load list` or `BLOB to list` commands. You can display this list in a floating palette; when each user clicks on a keyword in the list, the item chosen is inserted into the enterable area that is selected in the foreground process. The important thing is that you only process the item selected, because the `Selected list items` command returns the position of the item that you must process. When using this position value, you obtain the title of the item by means of the `GET LIST ITEM` command. Here again, you do not need to identify each item individually; you can pass any value (except 0) in the *itemRef* parameter.
2. You need to partially identify the list items (intermediary level).
 

You use the item reference number to store information needed when you must work with the item; this point is detailed in the example of the `APPEND TO LIST` command. In this example, we use the item reference numbers to store record numbers. However, we must be able to establish a distinction between items that correspond to the [Department] records and those that correspond to the [Employees] records.
3. You need to identify all the list items individually (advanced level).
 

You program an elaborate management of hierarchical lists in which you absolutely must be able to identify each item individually at every level of the list. A simple way of implementing this is to maintain a personal counter. Suppose that you create a *hList* list using the `APPEND TO LIST` command. At this stage, you initialize a counter *vhlCounter* to 1. Each time you call `APPEND TO LIST` or `INSERT IN LIST`, you increment this counter (`vhlCounter:=vhlCounter+1`), and you pass the counter number as the item reference number. The trick consists in never decrementing the counter when you delete items — the counter can only increase. In this way, you guarantee the uniqueness of the item reference numbers. Since these numbers are of the Longint type, you can add or insert more than two billion items in a list that has been reinitialized... (however if

you are working with such a great number of items, this usually means that you should use a table rather than a list.)

If you use Bitwise Operators, you can also use item reference numbers for storing information that can be put into a Longint, i.e. 2 Integers, 4-byte values or, yet again, 32 Booleans.

## When do you need unique reference numbers?

In most cases, when using hierarchical lists for user interface purposes and when only dealing with the selected item (the one that was clicked or dragged), you will not need to use item reference numbers at all. Using `Selected list items` and `GET LIST ITEM` you have all you need to deal with the currently selected item. In addition, commands such as `INSERT IN LIST` and `DELETE FROM LIST` allow you to manipulate the list "relatively" with respect to the selected item.

Basically, you need to deal with item reference numbers when you want direct access to any item of the list programmatically and not necessarily the one currently selected in the list.

## Modifiable element

You can control whether hierarchical list items can be modified by the user by using the **Alt+click**(Windows) / **Option+click** (macOS) shortcut, or by carrying out a long click on the text of the item.

- Whatever the hierarchical list data source, you can control the whole object with the [Enterable](#) property.
- In addition, if you populate the hierarchical list using a list created in the Lists editor, you control whether an item in a hierarchical list is modifiable using the **Modifiable Element** option in the Lists editor. For more information, see [Setting list properties](#).

## Supported Properties

[Bold](#) - [Border Line Style](#) - [Bottom](#) - [Choice List](#) - [Class](#) - [Draggable](#) - [Droppable](#) - [Enterable](#) - [Entry Filter](#) - [Fill Color](#) - [Focusable](#) - [Font](#) - [Font Color](#) - [Font Size](#) - [Height](#) - [Help Tip](#) - [Hide focus rectangle](#) - [Horizontal Scroll Bar](#) - [Horizontal Sizing](#) - [Italic](#) - [Left](#) - [Multi-selectable](#) - [Object Name](#) - [Right](#) - [Top](#) - [Type](#) - [Underline](#) - [Vertical Scroll Bar](#) - [Vertical Sizing](#) - [Variable or Expression](#) - [Visibility](#) - [Width](#)

## List Box

List boxes are complex active objects that allow displaying and entering data as synchronized columns. They can be bound to database contents such as entity selections and record sections, or to any language contents such as collections and arrays. They include advanced features regarding data entry, column sorting, event management, customized appearance, moving of columns, etc.

A list box contains one or more columns whose contents are automatically synchronized. The number of columns is, in theory, unlimited (it depends on the machine resources).

## Overview

### Basic user features

During execution, list boxes allow displaying and entering data as lists. To make a cell editable ([if entry is allowed for the column](#)), simply click twice on the value that it contains:

| Last name | First name |
|-----------|------------|
| James     | Henry      |
| Jameson   | Marc       |

Users can enter and display the text on several lines within a list box cell. To add a line break, press **Ctrl+Carriage return** on Windows or **Option+Carriage return** on macOS.

Booleans and pictures can be displayed in cells, as well as dates, times, or numbers. It is possible to sort column values by clicking on a header ([standard sort](#)). All columns are automatically synchronized.

It is also possible to resize each column, and the user can modify the order of [columns](#) and [rows](#) by moving them using the mouse, if this action is authorized. Note that list boxes can be used in [hierarchical mode](#).

The user can select one or more rows using the standard shortcuts: **Shift+click** for an adjacent selection and **Ctrl+click** (Windows) or **Command+click** (macOS) for a non-adjacent selection.

### List box parts

A list box is composed of four distinct parts:

- the list box object in its entirety,
- columns,
- column headers, and
- column footers.

Each part has its own name as well as specific properties. For example, the number of columns or the alternating color of each row is set in the list box object properties, the width of each column is set in the column properties, and the font of the header is set

in the header properties.

It is possible to add an object method to the list box object and/or to each column of the list box. Object methods are called in the following order:

1. Object method of each column
2. Object method of the list box

The column object method gets events that occur in its [header](#) and [footer](#).

## List box types

There are several types of list boxes, with their own specific behaviors and properties. The list box type depends on its [Data Source property](#):

- **Arrays:** each column is bound to a 4D array. Array-based list boxes can be displayed as [hierarchical list boxes](#).
- **Selection (Current selection or Named selection):** each column is bound to an expression (e.g. a field) which is evaluated for every record of the selection.
- **Collection or Entity selection:** each column is bound to an expression which is evaluated for every element of the collection or every entity of the entity selection.

It is not possible to combine different list box types in the same list box object. The data source is set when the list box is created. It is then no longer possible to modify it by programming.

## Managing list boxes

You can completely configure a list box object through its properties, and you can also manage it dynamically through programming.

The 4D Language includes a dedicated "List Box" theme for list box commands, but commands from various other themes, such as "Object properties" commands or `EDIT ITEM`, `Displayed line number` commands can also be used. Refer to the [List Box Commands Summary](#) page of the *4D Language reference* for more information.

## List box objects

### Array list boxes

In an array list box, each column must be associated with a one-dimensional 4D array; all array types can be used, with the exception of pointer arrays. The number of rows is based on the number of array elements.

By default, 4D assigns the name "ColumnX" to each column. You can change it, as well as other column properties, in the [column properties](#). The display format for each column can also be defined using the `OBJECT SET FORMAT` command.

Array type list boxes can be displayed in [hierarchical mode](#), with specific mechanisms.

With array type list box, the values entered or displayed are managed using the 4D language. You can also associate a [choice list](#) with a column in order to control data entry. The values of columns are managed using high-level List box commands (such as `LISTBOX INSERT ROWS` or `LISTBOX DELETE ROWS`) as well as array manipulation commands. For example, to initialize the contents of a column, you can use the following instruction:

```
ARRAY TEXT(varCol;size)
```

You can also use a list:

```
LIST TO ARRAY("ListName";varCol)
```

**Warning:** When a list box contains several columns of different sizes, only the number of items of the smallest array (column) will be displayed. You should make sure that each array has the same number of elements as the others. Also, if a list box column is empty (this occurs when the associated array was not correctly declared or sized using the language), the list box displays nothing.

## Selection list boxes

In this type of list box, each column can be associated with a field (for example `[Employees] LastName`) or an expression. The expression can be based on one or more fields (for example, `[Employees] FirstName + " " [Employees] LastName`) or it may simply be a formula (for example `String(Milliseconds)`). The expression can also be a project method, a variable or an array item. You can use the `LISTBOX SET COLUMN FORMULA` and `LISTBOX INSERT COLUMN FORMULA` commands to modify columns programmatically.

The contents of each row is then evaluated according to a selection of records: the **current selection** of a table or a **named selection**.

In the case of a list box based on the current selection of a table, any modification done from the database side is automatically reflected in the list box, and vice versa. The current selection is therefore always the same in both places.

## Collection or Entity selection list boxes

In this type of list box, each column must be associated to an expression. The contents of each row is then evaluated per collection element or per entity of the entity selection.

Each element of the collection or each entity is available as an object that can be accessed through the `This` keyword. A column expression can be a property path, a project method, a variable, or any formula, accessing each entity or collection element object through `This`, for example `This.<propertyPath>` (or `This.value` in case of a collection of scalar values). You can use the `LISTBOX SET COLUMN FORMULA` and `LISTBOX INSERT COLUMN FORMULA` commands to modify columns programmatically.

When the data source is an entity selection, any modifications made on the list box side are automatically saved in the database. On the other hand, modifications made on the database side are visible in the list box after touched entities have been reloaded.

When the data source is a collection, any modifications made in the list box values are reflected in the collection. On the other hand, if modifications are done on the collection using for example the various functions of the [Collection class](#), you will need to explicitly notify 4D by reassigning the collection variable to itself, so that the list box contents is refreshed. For example:

```
myCol:=myCol.push("new value") //display new value in list box
```

## Supported Properties

Supported properties depend on the list box type.

| Property                                       | Array list<br>box | Selection list<br>box | Collection or Entity Selection<br>list box |
|------------------------------------------------|-------------------|-----------------------|--------------------------------------------|
| <a href="#">Alternate Background Color</a>     | X                 | X                     | X                                          |
| <a href="#">Background Color</a>               | X                 | X                     | X                                          |
| <a href="#">Bold</a>                           | X                 | X                     | X                                          |
| <a href="#">Background Color Expression</a>    |                   | X                     | X                                          |
| <a href="#">Border Line Style</a>              | X                 | X                     | X                                          |
| <a href="#">Bottom</a>                         | X                 | X                     | X                                          |
| <a href="#">Class</a>                          | X                 | X                     | X                                          |
| <a href="#">Collection or entity selection</a> |                   | X                     | X                                          |
| <a href="#">Column Auto-Resizing</a>           | X                 | X                     | X                                          |
| <a href="#">Current item</a>                   |                   |                       | X                                          |
| <a href="#">Current item position</a>          |                   |                       | X                                          |
| <a href="#">Data Source</a>                    | X                 | X                     | X                                          |
| <a href="#">Detail Form Name</a>               |                   | X                     |                                            |
| <a href="#">Display Headers</a>                | X                 | X                     | X                                          |
| <a href="#">Display Footers</a>                | X                 | X                     | X                                          |
| <a href="#">Double-click on row</a>            |                   | X                     |                                            |
| <a href="#">Draggable</a>                      | X                 | X                     | X                                          |
| <a href="#">Droppable</a>                      | X                 | X                     | X                                          |
| <a href="#">Focusable</a>                      | X                 | X                     | X                                          |
| <a href="#">Font</a>                           | X                 | X                     | X                                          |
| <a href="#">Font Color</a>                     | X                 | X                     | X                                          |
| <a href="#">Font Color Expression</a>          |                   | X                     | X                                          |
| <a href="#">Font Size</a>                      | X                 | X                     | X                                          |
| <a href="#">Height (list box)</a>              | X                 | X                     | X                                          |
| <a href="#">Height (headers)</a>               | X                 | X                     | X                                          |
| <a href="#">Height (footers)</a>               | X                 | X                     | X                                          |
| <a href="#">Hide extra blank rows</a>          | X                 | X                     | X                                          |
| <a href="#">Hide focus rectangle</a>           | X                 | X                     | X                                          |
| <a href="#">Hide selection highlight</a>       | X                 | X                     | X                                          |
| <a href="#">Hierarchical List Box</a>          | X                 |                       |                                            |
| <a href="#">Highlight Set</a>                  |                   | X                     |                                            |
| <a href="#">Horizontal Alignment</a>           | X                 | X                     | X                                          |
| <a href="#">Horizontal Line Color</a>          | X                 | X                     | X                                          |
| <a href="#">Horizontal Padding</a>             | X                 | X                     | X                                          |
| <a href="#">Horizontal Scroll Bar</a>          | X                 | X                     | X                                          |
| <a href="#">Horizontal Sizing</a>              | X                 | X                     | X                                          |
| <a href="#">Italic</a>                         | X                 | X                     | X                                          |
| <a href="#">Left</a>                           | X                 | X                     | X                                          |
| <a href="#">Master Table</a>                   |                   | X                     |                                            |
| <a href="#">Meta info expression</a>           |                   |                       | X                                          |
| <a href="#">Method</a>                         | X                 | X                     | X                                          |
| <a href="#">Movable Rows</a>                   | X                 |                       |                                            |
| <a href="#">Named Selection</a>                |                   | X                     |                                            |
| <a href="#">Number of Columns</a>              | X                 | X                     | X                                          |
| <a href="#">Number of Locked Columns</a>       | X                 | X                     | X                                          |
| <a href="#">Number of Static Columns</a>       | X                 | X                     | X                                          |

| <u>Object Name</u>            | <u>Property</u> | x | Array list | x | Selection list | x | Collection or Entity Selection |
|-------------------------------|-----------------|---|------------|---|----------------|---|--------------------------------|
|                               |                 |   | box        |   | box            |   | list box                       |
| <u>Right</u>                  |                 | X |            |   |                | X |                                |
| <u>Row Background Color</u>   |                 | X |            |   |                |   |                                |
| <u>Array</u>                  |                 |   |            |   |                |   |                                |
| <u>Row Control Array</u>      |                 | X |            |   |                |   |                                |
| <u>Row Font Color Array</u>   |                 | X |            |   |                |   |                                |
| <u>Row Height</u>             |                 | X |            |   |                |   |                                |
| <u>Row Height Array</u>       |                 | X |            |   |                |   |                                |
| <u>Row Style Array</u>        |                 | X |            |   |                |   |                                |
| <u>Selected Items</u>         |                 |   |            |   |                | X |                                |
| <u>Selection Mode</u>         |                 | X |            | X |                | X |                                |
| <u>Single-Click Edit</u>      |                 | X |            | X |                | X |                                |
| <u>Sortable</u>               |                 | X |            | X |                | X |                                |
| <u>Standard action</u>        |                 | X |            |   |                |   |                                |
| <u>Style Expression</u>       |                 |   |            | X |                | X |                                |
| <u>Top</u>                    |                 | X |            | X |                | X |                                |
| <u>Transparent</u>            |                 | X |            | X |                | X |                                |
| <u>Type</u>                   |                 | X |            | X |                | X |                                |
| <u>Underline</u>              |                 | X |            | X |                | X |                                |
| <u>Variable or Expression</u> |                 | X |            | X |                |   |                                |
| <u>Vertical Alignment</u>     |                 | X |            | X |                | X |                                |
| <u>Vertical Line Color</u>    |                 | X |            | X |                | X |                                |
| <u>Vertical Padding</u>       |                 | X |            | X |                | X |                                |
| <u>Vertical Scroll Bar</u>    |                 | X |            | X |                | X |                                |
| <u>Vertical Sizing</u>        |                 | X |            | X |                | X |                                |
| <u>Visibility</u>             |                 | X |            | X |                | X |                                |
| <u>Width</u>                  |                 | X |            | X |                | X |                                |

List box columns, headers and footers support specific properties.

## Supported Form Events

| Form event           | Additional Properties Returned<br>(see <a href="#">Form event</a> for main properties)                                                                 | Comments                                                                                     |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|
| On After Edit        | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> </ul>        |                                                                                              |
| On After Keystroke   | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> </ul>        |                                                                                              |
| On After Sort        | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">headerName</a></li> </ul> | <p><i>Compound formulas cannot be sorted.<br/>(e.g., This.firstName + This.lastName)</i></p> |
| On Alternative Click | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> </ul>        | <p><i>Arrays list boxes only</i></p>                                                         |
| On Before Data Entry | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> </ul>        |                                                                                              |
| On Before Keystroke  | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> </ul>        |                                                                                              |

|                               |                                                                                                                                                                                                                                        | Comments                                                                          |
|-------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|
| On Begin Drag Over Form Event | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a> (see <a href="#">Form event</a> for main properties)</li> <li>• <a href="#">row</a></li> <li>• <a href="#">column</a></li> </ul> |                                                                                   |
| On Clicked                    | <ul style="list-style-type: none"> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> </ul>                                                                                                                          |                                                                                   |
| On Close Detail               | <ul style="list-style-type: none"> <li>• <a href="#">row</a></li> </ul>                                                                                                                                                                | <i>Current Selection &amp; Named Selection list boxes only</i>                    |
| On Collapse                   | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> </ul>                                                                                        | <i>Hierarchical list box only</i>                                                 |
| On Column Moved               | <ul style="list-style-type: none"> <li>• <a href="#">columnName</a></li> <li>• <a href="#">newPosition</a></li> <li>• <a href="#">oldPosition</a></li> </ul>                                                                           |                                                                                   |
| On Column Resize              | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">newSize</a></li> <li>• <a href="#">oldSize</a></li> </ul>                                                 |                                                                                   |
| On Data Change                | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> </ul>                                                                                        |                                                                                   |
| On Delete Action              | <ul style="list-style-type: none"> <li>• <a href="#">row</a></li> </ul>                                                                                                                                                                |                                                                                   |
| On Display Detail             | <ul style="list-style-type: none"> <li>• <a href="#">isRowSelected</a></li> <li>• <a href="#">row</a></li> </ul>                                                                                                                       |                                                                                   |
| On Double Clicked             | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> <li>• <a href="#">area</a></li> </ul>                                                        |                                                                                   |
| On Drag Over                  | <ul style="list-style-type: none"> <li>• <a href="#">areaName</a></li> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> <li>• <a href="#">column</a></li> </ul>                  |                                                                                   |
| On Drop                       | <ul style="list-style-type: none"> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> </ul>                                                                                                                          |                                                                                   |
| On Expand                     | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> </ul>                                                                                        | <i>Hierarchical list box only</i>                                                 |
| On Footer Click               | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">footerName</a></li> </ul>                                                                                 | <i>Arrays, Current Selection &amp; Named Selection list boxes only</i>            |
| On Getting Focus              | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> </ul>                                                                                        | <i>Additional properties returned only when editing a cell</i>                    |
| On Header Click               | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">headerName</a></li> </ul>                                                                                 |                                                                                   |
| On Load                       |                                                                                                                                                                                                                                        |                                                                                   |
| On Losing Focus               | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> <li>• <a href="#">area</a></li> </ul>                                                        | <i>Additional properties returned only when editing a cell has been completed</i> |
| On Mouse Enter                | <ul style="list-style-type: none"> <li>• <a href="#">areaName</a></li> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> </ul>                                                                                   |                                                                                   |

| <b>Form Event</b>   | <b>Additional Properties Returned</b>                                         | <b>Comments</b>                                                |
|---------------------|-------------------------------------------------------------------------------|----------------------------------------------------------------|
| On Mouse Leave      | • <a href="#">row</a><br>(see <a href="#">Form event</a> for main properties) |                                                                |
| On Mouse Move       | • <a href="#">area</a><br>• <a href="#">areaName</a>                          |                                                                |
|                     | • <a href="#">column</a><br>• <a href="#">columnName</a>                      |                                                                |
|                     | • <a href="#">row</a>                                                         |                                                                |
| On Open Detail      | • <a href="#">row</a>                                                         | <i>Current Selection &amp; Named Selection list boxes only</i> |
| On Row Moved        | • <a href="#">newPosition</a><br>• <a href="#">oldPosition</a>                | <i>Arrays list boxes only</i>                                  |
| On Selection Change |                                                                               |                                                                |
| On Scroll           | • <a href="#">horizontalScroll</a><br>• <a href="#">verticalScroll</a>        |                                                                |
| On Unload           |                                                                               |                                                                |

## Additional Properties

Form events on list box or list box column objects may return the following additional properties:

| <b>Property</b>  | <b>Type</b> | <b>Description</b>                                                    |
|------------------|-------------|-----------------------------------------------------------------------|
| area             | text        | List box object area ("header", "footer", "cell")                     |
| areaName         | text        | Name of the area                                                      |
| column           | longint     | Column number                                                         |
| columnName       | text        | Name of the column                                                    |
| footerName       | text        | Name of the footer                                                    |
| headerName       | text        | Name of the header                                                    |
| horizontalScroll | longint     | Positive if scroll is towards the right, negative if towards the left |
| isRowSelected    | boolean     | True if row is selected, else False                                   |
| newPosition      | longint     | New position of the column or row                                     |
| newSize          | longint     | New size (in pixels) of the column or row                             |
| oldPosition      | longint     | Previous position of the column or row                                |
| oldSize          | longint     | Previous size (in pixels) of the column or row                        |
| row              | longint     | Row number                                                            |
| verticalScroll   | longint     | Positive if scroll is towards the bottom, negative if towards the top |

If an event occurs on a "fake" column or row that doesn't exist, an empty string is typically returned.

## List box columns

A list box is made of one or more column object(s) which have specific properties. You can select a list box column in the Form editor by clicking on it when the list box object is selected:

You can set standard properties (text, background color, etc.) for each column of the list box; these properties take priority over those of the list box object properties.

You can define the [Expression type](#) for array list box columns (String, Text, Number, Date, Time, Picture, Boolean, or Object).

## Column Specific Properties

[Alpha Format](#) - [Alternate Background Color](#) - [Automatic Row Height](#) - [Background Color](#) - [Background Color Expression](#) - [Bold](#) - [Choice List](#) - [Class](#) - [Data Type \(selection and collection list box column\)](#) - [Date Format](#) - [Default Values](#) - [Display Type](#) - [Enterable](#) - [Entry Filter](#) - [Excluded List](#) - [Expression](#) - [Expression Type \(array list box column\)](#) - [Font](#) - [Font Color](#) - [Horizontal Alignment](#) - [Horizontal Padding](#) - [Italic](#) - [Invisible](#) - [Maximum Width](#) - [Method](#) - [Minimum Width](#) - [Multi-style](#) - [Number Format](#) - [Object Name](#) - [Picture Format](#) - [Resizable](#) - [Required List](#) - [Row Background Color](#) - [Array](#) - [Row Font Color Array](#) - [Row Style Array](#) - [Save as](#) - [Style Expression](#) - [Text when False/Text when True](#) - [Time Format](#) - [Truncate with ellipsis](#) - [Underline](#) - [Variable or Expression](#) - [Vertical Alignment](#) - [Vertical Padding](#) - [Width](#) - [Wordwrap](#)

## Supported Form Events

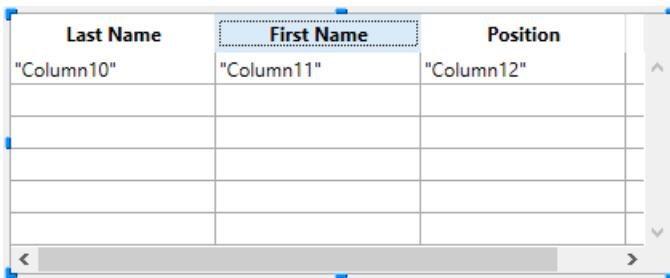
| Form event           | Additional Properties Returned<br>(see <a href="#">Form event</a> for main properties)                                                                                                 | Comments                                                                              |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| On After Edit        | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> </ul>                                        |                                                                                       |
| On After Keystroke   | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> </ul>                                        |                                                                                       |
| On After Sort        | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">headerName</a></li> </ul>                                 | <i>Compound formulas cannot be sorted.<br/>(e.g., This.firstName + This.lastName)</i> |
| On Alternative Click | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> </ul>                                        | <i>Arrays list boxes only</i>                                                         |
| On Before Data Entry | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> </ul>                                        |                                                                                       |
| On Before Keystroke  | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> </ul>                                        |                                                                                       |
| On Begin Drag Over   | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> <li>• <a href="#">column</a></li> </ul>      |                                                                                       |
| On Clicked           | <ul style="list-style-type: none"> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> </ul>                                                                          |                                                                                       |
| On Column Moved      | <ul style="list-style-type: none"> <li>• <a href="#">columnName</a></li> <li>• <a href="#">newPosition</a></li> <li>• <a href="#">oldPosition</a></li> </ul>                           |                                                                                       |
| On Column Resize     | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">newSize</a></li> <li>• <a href="#">oldSize</a></li> </ul> |                                                                                       |
| On Data Change       | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> <li>• <a href="#">column</a></li> </ul>      |                                                                                       |

|                              |                                                                                                                                                                                                                       | Comments                                                                   |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------|
| On Double Clicked Form event | <ul style="list-style-type: none"> <li>• <a href="#">columnName</a> (see <a href="#">Form event</a> for main properties)</li> <li>• <a href="#">row</a></li> <li>• <a href="#">area</a></li> </ul>                    |                                                                            |
| On Drag Over                 | <ul style="list-style-type: none"> <li>• <a href="#">areaName</a></li> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> <li>• <a href="#">column</a></li> </ul> |                                                                            |
| On Drop                      | <ul style="list-style-type: none"> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> </ul>                                                                                                         |                                                                            |
| On Footer Click              | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">footerName</a></li> </ul>                                                                | Arrays, Current Selection & Named Selection list boxes only                |
| On Getting Focus             | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> </ul>                                                                       | Additional properties returned only when editing a cell                    |
| On Header Click              | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">headerName</a></li> </ul>                                                                |                                                                            |
| On Load                      |                                                                                                                                                                                                                       |                                                                            |
| On Losing Focus              | <ul style="list-style-type: none"> <li>• <a href="#">column</a></li> <li>• <a href="#">columnName</a></li> <li>• <a href="#">row</a></li> </ul>                                                                       | Additional properties returned only when editing a cell has been completed |
| On Row Moved                 | <ul style="list-style-type: none"> <li>• <a href="#">newPosition</a></li> <li>• <a href="#">oldPosition</a></li> </ul>                                                                                                | Arrays list boxes only                                                     |
| On Scroll                    | <ul style="list-style-type: none"> <li>• <a href="#">horizontalScroll</a></li> <li>• <a href="#">verticalScroll</a></li> </ul>                                                                                        |                                                                            |
| On Unload                    |                                                                                                                                                                                                                       |                                                                            |

## List box headers

To be able to access the header properties of a list box, you must enable the [Display Headers](#) option of the list box.

When headers are displayed, you can select a header in the Form editor by clicking it when the list box object is selected:



You can set standard text properties for each column header of the list box; in this case, these properties have priority over those of the column or of the list box itself.

In addition, you have access to the specific properties for headers. Specifically, an icon can be displayed in the header next to or in place of the column title, for example when performing [customized sorts](#).

|       |      |               |
|-------|------|---------------|
|       | Name | Position      |
| Smith |      | Administrator |

At runtime, events that occur in a header are generated in the [list box column object method](#).

When the `OBJECT SET VISIBLE` command is used with a header, it is applied to all headers, regardless of the individual element set by the command. For example, `OBJECT SET VISIBLE(*;"header3";False)` will hide all headers in the list box object to which *header3* belongs and not simply this header.

## Header Specific Properties

[Bold](#) - [Class](#) - [Font](#) - [Font Color](#) - [Help Tip](#) - [Horizontal Alignment](#) - [Horizontal Padding](#) - [Icon Location](#) - [Italic](#) - [Object Name](#) - [Pathname](#) - [Title](#) - [Underline](#) - [Variable or Expression](#) - [Vertical Alignment](#) - [Vertical Padding](#) - [Width](#)

## List box footers

To be able to access footer properties for a list box, you must enable the [Display Footers](#) option.

List boxes can contain non-enterable "footers" displaying additional information. For data shown in table form, footers are usually used to display calculations such as totals or averages.

When footers are displayed, you can click to select one when the list box object is selected in the Form editor:

For each List box column footer, you can set standard text properties: in this case, these properties take priority over those of the column or of the list box. You can also access specific properties for footers. In particular, you can insert a [custom or automatic calculation](#).

At runtime, events that occur in a footer are generated in the [list box column object method](#).

When the `OBJECT SET VISIBLE` command is used with a footer, it is applied to all footers, regardless of the individual element set by the command. For example, `OBJECT SET VISIBLE(*;"footer3";False)` will hide all footers in the list box object to which *footer3* belongs and not simply this footer.

## Footer Specific Properties

[Alpha Format](#) - [Background Color](#) - [Bold](#) - [Class](#) - [Date Format](#) - [Expression Type](#) - [Font](#) - [Font Color](#) - [Help Tip](#) - [Horizontal Alignment](#) - [Horizontal Padding](#) - [Italic](#) - [Number Format](#) - [Object Name](#) - [Picture Format](#) - [Time Format](#) - [Truncate with ellipsis](#) - [Underline](#) - [Variable Calculation](#) - [Variable or Expression](#) - [Vertical Alignment](#) - [Vertical Padding](#) - [Width](#) - [Wordwrap](#)

## Managing entry

For a list box cell to be enterable, both of the following conditions must be met:

- The cell's column must have been set as [Enterable](#) (otherwise, the cells of the column can never be enterable).
- In the `On Before Data Entry` event, \$0 does not return -1. When the cursor arrives in the cell, the `On Before Data Entry` event is generated in the column method. If, in the context of this event, \$0 is set to -1, the cell is considered as

not enterable. If the event was generated after **Tab** or **Shift+Tab** was pressed, the focus goes to either the next cell or the previous one, respectively. If \$0 is not -1 (by default \$0 is 0), the cell is enterable and switches to editing mode.

Let's consider the example of a list box containing two arrays, one date and one text. The date array is not enterable but the text array is enterable if the date has not already past.

| Header1              | Header2              |
|----------------------|----------------------|
| Variable Name: tDate | Variable Name: tText |
|                      |                      |
|                      |                      |
|                      |                      |
|                      |                      |
|                      |                      |

Here is the method of the *arrText* column:

```
Case of
  :(FORM event.code=On Before Data Entry) // a cell gets the focus
    LISTBOX GET CELL POSITION(*;"lb";$col;$row)
  // identification of cell
    If(arrDate{$row}<Current date) // if date is earlier than today
      $0:=-1 // cell is NOT enterable
    Else
      // otherwise, cell is enterable
      End if
End case
```

The `On Before Data Entry` event is returned before `On Getting Focus`.

In order to preserve data consistency for selection type and entity selection type list boxes, any modified record/entity is automatically saved as soon as the cell is validated, i.e.:

- when the cell is deactivated (user presses tab, clicks, etc.)
- when the listbox is no longer focused,
- when the form is no longer focused.

The typical sequence of events generated during data entry or modification is as follows:

| Action                                                                                     | Listbox type(s)                                                                                  | Sequence of events                                                                                                                                                                                                                    |
|--------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| A cell switches to edit mode (user action or a call to the <code>EDIT ITEM</code> command) | All                                                                                              | On Before Data Entry                                                                                                                                                                                                                  |
| Its value is modified                                                                      | All                                                                                              | On Getting Focus                                                                                                                                                                                                                      |
|                                                                                            | All                                                                                              | On Before Keystroke                                                                                                                                                                                                                   |
|                                                                                            | All                                                                                              | On After Keystroke                                                                                                                                                                                                                    |
|                                                                                            | All                                                                                              | On After Edit                                                                                                                                                                                                                         |
| A user validates and leaves the cell                                                       | Selection<br>list<br>boxes<br>Record<br>selection<br>list<br>boxes<br>Selection<br>list<br>boxes | On saving an existing record trigger (if set)                                                                                                                                                                                         |
|                                                                                            | Entity<br>list<br>boxes                                                                          | Entity is saved with automerge option, optimistic lock selection (see <code>entity.save()</code> ). In case of successful save, the entity is refreshed with the last update done. If the save operation fails, an error is displayed |
|                                                                                            | All                                                                                              | On Losing Focus                                                                                                                                                                                                                       |

(\*) With entity selection list boxes, in the On Data Change event:

- the [Current item](#) object contains the value before modification.
- the [This](#) object contains the modified value.

Data entry in collection/entity selection type list boxes has a limitation when the expression evaluates to null. In this case, it is not possible to edit or remove the null value in the cell.

## Managing selections

Selections are managed differently depending on whether the list box is based on an array, on a selection of records, or on a collection/entity selection:

- **Selection list box:** Selections are managed by a set, which you can modify if necessary, called `$ListboxSetX` by default (where X starts at 0 and is incremented based on the number of list boxes in the form). This set is [defined in the properties](#) of the list box. It is automatically maintained by 4D: If the user selects one or more rows in the list box, the set is immediately updated. On the other hand, it is also possible to use the commands of the "Sets" theme in order to modify the selection of the list box via programming.
- **Collection/Entity selection list box:** Selections are managed through dedicated list box properties:
  - [Current item](#) is an object that will receive the selected element/entity
  - [Selected Items](#) is a collection of selected items
  - [Current item position](#) returns the position of the selected element or entity.
- **Array list box:** The `LISTBOX SELECT ROW` command can be used to select one or more rows of the list box by programming. The [variable linked to the List box](#)

[object](#) is used to get, set or store selections of object rows. This variable corresponds to a Boolean array that is automatically created and maintained by 4D. The size of this array is determined by the size of the list box: it contains the same number of elements as the smallest array linked to the columns. Each element of this array contains `True` if the corresponding line is selected and `False` otherwise. 4D updates the contents of this array depending on user actions. Inversely, you can change the value of array elements to change the selection in the list box. On the other hand, you can neither insert nor delete rows in this array; you cannot retype rows either. The `Count in array` command can be used to find out the number of selected lines. For example, this method allows inverting the selection of the first row of the (array type) list box:

```
ARRAY BOOLEAN(tBLListBox;10)
//tBLListBox is the name of the list box variable in the form
If(tBLListBox{1}=True)
    tBLListBox{1}:=False
Else
    tBLListBox{1}:=True
End if
```

The `OBJECT SET SCROLL POSITION` command scrolls the list box rows so that the first selected row or a specified row is displayed.

## Customizing appearance of selected rows

When the [Hide selection highlight](#) option is selected, you need to make list box selections visible using available interface options. Since selections are still fully managed by 4D, this means:

- For array type list boxes, you must parse the Boolean array variable associated with the list box to determine which rows are selected or not.
- For selection type list boxes, you have to check whether the current record (row) belongs to the set specified in the [Highlight Set](#) property of the list box.

You can then define specific background colors, font colors and/or font styles by programming to customize the appearance of selected rows. This can be done using arrays or expressions, depending on the type of list box being displayed (see the following sections).

You can use the `lk inherited` constant to mimic the current appearance of the list box (e.g., font color, background color, font style, etc.).

## Selection list boxes

To determine which rows are selected, you have to check whether they are included in the set indicated in the [Highlight Set](#) property of the list box. You can then define the appearance of selected rows using one or more of the relevant [color or style expression property](#).

Keep in mind that expressions are automatically re-evaluated each time the:

- list box selection changes.
- list box gets or loses the focus.
- form window containing the list box becomes, or ceases to be, the frontmost window.

## Array list boxes

You have to parse the Boolean array [Variable or Expression](#) associated with the list box

to determine whether rows are selected or not selected.

You can then define the appearance of selected rows using one or more of the relevant [color or style array property](#).

Note that list box arrays used for defining the appearance of selected rows must be recalculated during the `On Selection Change` form event; however, you can also modify these arrays based on the following additional form events:

- `On Getting Focus` (list box property)
- `On Losing Focus` (list box property)
- `On Activate` (form property)
- `On Deactivate` (form property) ...depending on whether and how you want to visually represent changes of focus in selections.

## Example

You have chosen to hide the system highlight and want to display list box selections with a green background color, as shown here:

For an array type list box, you need to update the [Row Background Color Array](#) by programming. In the JSON form, you have defined the following Row Background Color Array for the list box:

```
"rowFillSource": "_ListboxBackground",
```

In the object method of the list box, you can write:

```
Case of
  :(FORM event.code=On Selection Change)
    $n:=Size of array(LB_Arrays)
    ARRAY LONGINT(_ListboxBackground;$n) // row background colors
    For($i;1;$n)
      If(LB_Arrays{$i}=True) // selected
        _ListboxBackground{$i}:=0x0080C080 // green background
      Else // not selected
        _ListboxBackground{$i}:=lk inherited
      End if
    End for
End case
```

For a selection type list box, to produce the same effect you can use a method to update the [Background Color Expression](#) based on the set specified in the [Highlight Set](#) property.

For example, in the JSON form, you have defined the following Highlight Set and Background Color Expression for the list box:

```
"highlightSet": "$$SampleSet",
"rowFillSource": "UI_SetColor",
```

You can write in the `UI_SetColor` method:

```

If(Is in set("$SampleSet"))
    $color:=0x0080C080 // green background
Else
    $color:=lk inherited
End if

$0:=$color

```

In hierarchical list boxes, break rows cannot be highlighted when the [Hide selection highlight](#) option is checked. Since it is not possible to have distinct colors for headers of the same level, there is no way to highlight a specific break row by programming.

## Managing sorts

A sort in a list box can be standard or custom. When a column of a list box is sorted, all other columns are always synchronized automatically.

### Standard sort

By default, a list box provides standard column sorts when the header is clicked. A standard sort is an alphanumeric sort of evaluated column values, alternately ascending/descending with each successive click.

You can enable or disable standard user sorts by disabling the [Sortable](#) property of the list box (enabled by default).

Standard sort support depends on the list box type:

|                                       | <b>Support</b> | <b>Comments</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------------------------------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>List box of type standard sort</b> |                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Collection of objects                 | Yes            | <ul style="list-style-type: none"> <li>"This.a" or "This.a.b" columns are sortable.</li> <li>The <a href="#">list box source property</a> must be an <a href="#">assignable expression</a>.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Collection of scalar values           | No             | Use custom sort with <a href="#">orderBy()</a> function <ul style="list-style-type: none"> <li>The <a href="#">list box source property</a> must be an <a href="#">assignable expression</a>.</li> <li>Supported: sorts on object attribute properties (e.g. "This.data.city" when "data" is an object attribute)</li> <li>Supported: sorts on related attributes (e.g. "This.company.name")</li> <li>Not supported: sorts on object attribute properties through related attributes (e.g. "This.company.data.city"). For this, you need to use custom sort with <a href="#">orderByFormula()</a> function (see example below)</li> </ul> |
| Entity selection                      | Yes            | Only simple expressions are sortable (e.g. [Table_1]Field_2) <ul style="list-style-type: none"> <li>Not supported: sorts on object attribute properties through related attributes (e.g. "This.company.data.city"). For this, you need to use custom sort with <a href="#">orderByFormula()</a> function (see example below)</li> </ul>                                                                                                                                                                                                                                                                                                   |
| Current selection                     | Yes            | Only simple expressions are sortable (e.g. [Table_1]Field_2)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| Named selection                       | No             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Arrays                                | Yes            | Columns bound to picture and pointer arrays are not sortable                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

### Custom sort

The developer can set up custom sorts, for example using the [LISTBOX\\_SORT\\_COLUMNS](#) command and/or combining the [On Header Click](#) and [On After Sort](#)

form events and relevant 4D commands.

Custom sorts allow you to:

- carry out multi-level sorts on several columns, thanks to the [LISTBOX SORT COLUMNS](#) command,
- use functions such as [collection.orderByFormula\(\)](#) or [entitySelection.orderByFormula\(\)](#) to sort columns on complex criteria.

## Example

You want to sort a list box using values of a property stored in a related object attribute. You have the following structure:

You design a list box of the entity selection type, bound to the `Form.child` expression. In the `On Load` form event, you execute `Form.child:=ds.Child.all()`.

You display two columns:

**Child name      Parent's nickname**

`This.name    This.parent.extra.nickname`

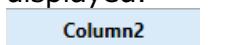
If you want to sort the list box using the values of the second column, you have to write:

```
If (Form event code=On Header Click)
```

```
Form.child:=Form.child.orderByFormula("This.parent.extra.nickname"; dk  
ascending)  
End if
```

## Column header variable

The value of the [column header variable](#) allows you to manage additional information: the current sort of the column (read) and the display of the sort arrow.

- If the variable is set to 0, the column is not sorted and the sort arrow is not displayed.  

- If the variable is set to 1, the column is sorted in ascending order and the sort arrow is displayed.  

- If the variable is set to 2, the column is sorted in descending order and the sort arrow is displayed.  


Only declared or dynamic [variables](#) can be used as header column variables. Other kinds of [expressions](#) such as `Form.sortValue` are not supported.

You can set the value of the variable (for example, `Header2:=2`) in order to "force" the sort arrow display. The column sort itself is not modified in this case; it is up to the developer to handle it.

The [OBJECT SET FORMAT](#) command offers specific support for icons in list box headers, which can be useful when you want to work with a customized sort icon.

# Managing row colors, styles, and display

There are several different ways to set background colors, font colors and font styles for list boxes:

- at the level of the [list box object properties](#),
- at the level of the [column properties](#),
- using [arrays or expressions properties](#) for the list box and/or for each column,
- at the level of the text of each cell (if [multi-style text](#)).

## Priority & inheritance

Priority and inheritance principles are observed when the same property is set at more than one level.

| Priority level | Setting location                                                                                                           |
|----------------|----------------------------------------------------------------------------------------------------------------------------|
| high priority  | Cell (if multi-style text)<br>Column arrays/methods<br>List box arrays/methods<br>Column properties<br>List box properties |
| low priority   | Meta Info expression (for collection or entity selection list boxes)                                                       |

For example, if you set a font style in the list box properties and another using a style array for the column, the latter one will be taken into account.

For each attribute (style, color and background color), an **inheritance** is implemented when the default value is used:

- for cell attributes: attribute values of rows
- for row attributes: attribute values of columns
- for column attributes: attribute values of the list box

This way, if you want an object to inherit the attribute value from a higher level, you can use pass the `lk_inherited` constant (default value) to the definition command or directly in the element of the corresponding style/color array. For example, given an array list box containing a standard font style with alternating colors:

|          |          |          |          |          |          |
|----------|----------|----------|----------|----------|----------|
| standard | standard | standard | standard | standard | standard |
| standard | standard | standard | standard | standard | standard |
| standard | standard | standard | standard | standard | standard |
| standard | standard | standard | standard | standard | standard |
| standard | standard | standard | standard | standard | standard |
| standard | standard | standard | standard | standard | standard |

You perform the following modifications:

- change the background of row 2 to red using the [Row Background Color Array](#) property of the list box object,
- change the style of row 4 to italics using the [Row Style Array](#) property of the list box object,
- two elements in column 5 are changed to bold using the [Row Style Array](#) property of the column 5 object,
- the 2 elements for column 1 and 2 are changed to dark blue using the [Row Background Color Array](#) property for the column 1 and 2 objects:

|          |          |          |          |          |          |
|----------|----------|----------|----------|----------|----------|
| standard | standard | standard | standard | standard | standard |
| standard | standard | standard | standard | standard | standard |
| standard | standard | standard | standard | standard | standard |
| standard | standard | standard | standard | standard | standard |
| standard | standard | standard | standard | standard | standard |

To restore the original appearance of the list box, you can:

- pass the `lk_inherited` constant in element 2 of the background color arrays for columns 1 and 2: then they inherit the red background color of the row.
- pass the `lk_inherited` constant in elements 3 and 4 of the style array for column 5: then they inherit the standard style, except for element 4, which changes to italics as specified in the style array of the list box.
- pass the `lk_inherited` constant in element 4 of the style array for the list box in order to remove the italics style.
- pass the `lk_inherited` constant in element 2 of the background color array for the list box in order to restore the original alternating color of the list box.

## Using arrays and expressions

Depending of the list box type, you can use different properties to customize row colors, styles and display:

| Property         | Array list box                             | Selection list box                                                                  | Collection or Entity Selection list box                                             |
|------------------|--------------------------------------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| Background color | <a href="#">Row Background Color Array</a> | <a href="#">Background Color Expression</a> or <a href="#">Meta info expression</a> | <a href="#">Background Color Expression</a> or <a href="#">Meta info expression</a> |
| Font color       | <a href="#">Row Font Color Array</a>       | <a href="#">Font Color Expression</a> or <a href="#">Meta info expression</a>       | <a href="#">Font Color Expression</a> or <a href="#">Meta info expression</a>       |
| Font style       | <a href="#">Row Style Array</a>            | <a href="#">Style Expression</a>                                                    | <a href="#">Style Expression</a> or <a href="#">Meta info expression</a>            |
| Display          | <a href="#">Row Control Array</a>          | -                                                                                   | -                                                                                   |

## Printing list boxes

Two printing modes are available: **preview mode** - which can be used to print a list box like a form object, and **advanced mode** - which lets you control the printing of the list box object itself within the form. Note that the "Printing" appearance is available for list box objects in the Form editor.

### Preview mode

Printing a list box in preview mode consists of directly printing the list box and the form that contains it using the standard print commands or the **Print** menu command. The list box is printed as it is in the form. This mode does not allow precise control of the printing of the object; in particular, it does not allow you to print all the rows of a list box that contains more rows than it can display.

### Advanced mode

In this mode, the printing of list boxes is carried out by programming, via the `Print object` command (project forms and table forms are supported). The `LISTBOX GET PRINT INFORMATION` command is used to control the printing of the object.

In this mode:

- The height of the list box object is automatically reduced when the number of

rows to be printed is less than the original height of the object (there are no "blank" rows printed). On the other hand, the height does not automatically increase according to the contents of the object. The size of the object actually printed can be obtained via the `LISTBOX GET PRINT INFORMATION` command.

- The list box object is printed "as is", in other words, taking its current display parameters into account: visibility of headers and gridlines, hidden and displayed rows, etc. These parameters also include the first row to be printed: if you call the `OBJECT SET SCROLL POSITION` command before launching the printing, the first row printed in the list box will be the one designated by the command.
- An automatic mechanism facilitates the printing of list boxes that contain more rows than it is possible to display: successive calls to `Print object` can be used to print a new set of rows each time. The `LISTBOX GET PRINT INFORMATION` command can be used to check the status of the printing while it is underway.

## Hierarchical list boxes

A hierarchical list box is a list box in which the contents of the first column appears in hierarchical form. This type of representation is adapted to the presentation of information that includes repeated values and/or values that are hierarchically dependent (country/region/city and so on).

Only [array type list boxes](#) can be hierarchical.

Hierarchical list boxes are a particular way of representing data but they do not modify the data structure (arrays). Hierarchical list boxes are managed exactly the same way as regular list boxes.

### Defining the hierarchy

To specify a hierarchical list box, there are several possibilities:

- Manually configure hierarchical elements using the Property list of the form editor (or edit the JSON form).
- Visually generate the hierarchy using the list box management pop-up menu, in the form editor.
- Use the [LISTBOX SET HIERARCHY](#) and [LISTBOX GET HIERARCHY](#) commands, described in the *4D Language Reference* manual.

### Hierarchical List Box property

This property specifies that the list box must be displayed in hierarchical form. In the JSON form, this feature is triggered [when the column `dataSource` property value is an array](#), i.e. a collection.

Additional options (**Variable 1...10**) are available when the *Hierarchical List Box* option is selected, corresponding to each element of the `dataSource` array to use as break column. Each time a value is entered in a field, a new row is added. Up to 10 variables can be specified. These variables set the hierarchical levels to be displayed in the first column.

The first variable always corresponds to the name of the variable for the first column of the list box (the two values are automatically bound). This first variable is always visible and enterable. For example: country. The second variable is also always visible and enterable; it specifies the second hierarchical level. For example: regions. Beginning with the third field, each variable depends on the one preceding it. For example: counties, cities, and so on. A maximum of ten hierarchical levels can be specified. If you remove a value, the whole hierarchy moves up a level.

The last variable is never hierarchical even if several identical values exists at this level. For example, referring to the configuration illustrated above, imagine that arr1 contains the values A A A B B B, arr2 has the values 1 1 1 2 2 2 and arr3 the values X X Y Y Y Z. In this case, A, B, 1 and 2 could appear in collapsed form, but not X and Y:

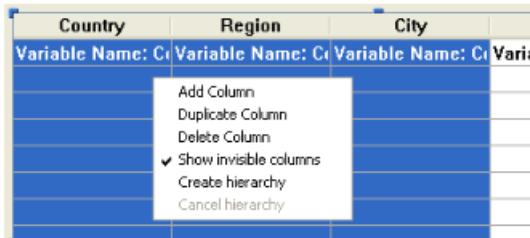
```
+ A
+ 1
  X
  X
  Y
+ B
+ 2
  Y
  Y
  Z
```

This principle is not applied when only one variable is specified in the hierarchy: in this case, identical values may be grouped.

If you specify a hierarchy based on the first columns of an existing list box, you must then remove or hide these columns (except for the first), otherwise they will appear in duplicate in the list box. If you specify the hierarchy via the pop-up menu of the editor (see below), the unnecessary columns are automatically removed from the list box.

### Create hierarchy using the contextual menu

When you select at least one column in addition to the first one in a list box object (of the array type) in the form editor, the **Create hierarchy** command is available in the context menu:



This command is a shortcut to define a hierarchy. When it is selected, the following actions are carried out:

- The **Hierarchical list box** option is checked for the object in the Property List.
- The variables of the columns are used to specify the hierarchy. They replace any variables already specified.
- The columns selected no longer appear in the list box (except for the title of the first one).

Example: given a list box whose first columns contain Country, Region, City and Population. When Country, Region and City are selected, if you choose **Create hierarchy** in the context menu, a three-level hierarchy is created in the first column, columns 2 and 3 are removed and the Population column becomes the second:

### Cancel hierarchy

When the first column is selected and already specified as hierarchical, you can use the **Cancel hierarchy** command. When you choose this command, the following actions are carried out:

- The **Hierarchical list box** option is deselected for the object,
  - The hierarchical levels 2 to X are removed and transformed into columns added to the list box.

## How it works

When a form containing a hierarchical list box is opened for the first time, by default all the rows are expanded.

A break row and a hierarchical "node" are automatically added in the list box when values are repeated in the arrays. For example, imagine a list box containing four arrays specifying cities, each city being characterized by its country, its region, its name and its number of inhabitants:

If this list box is displayed in hierarchical form (the first three arrays being included in the hierarchy), you obtain:

| City      | Population |
|-----------|------------|
| France    |            |
| Brittany  |            |
| Rennes    | 200000     |
| Quimper   | 80000      |
| Brest     | 120000     |
| Normandy  |            |
| Caen      | 75000      |
| Deauville | 35000      |
|           |            |
|           |            |

The arrays are not sorted before the hierarchy is constructed. If, for example, an array contains the data AAABBAACC, the hierarchy obtained is:

|   |   |
|---|---|
| > | A |
| > | B |
| > | A |
| > | C |

To expand or collapse a hierarchical "node," you can just click on it. If you **Alt+click** (Windows) or **Option+click** (macOS) on the node, all its sub-elements will be expanded or collapsed as well. These operations can also be carried out by programming using the `LISTBOX EXPAND` and `LISTBOX COLLAPSE` commands.

When values of the date or time type are included in a hierarchical list box, they are displayed in the short system format.

## Sorts in hierarchical list boxes

In a list box in hierarchical mode, a standard sort (carried out by clicking on the

header of a list box column) is always constructed as follows:

- In the first place, all the levels of the hierarchical column (first column) are automatically sorted by ascending order.
- The sort is then carried out by ascending or descending order (according to the user action) on the values of the column that was clicked.
- All the columns are synchronized.
- During subsequent sorts carried out on non-hierarchical columns of the list box, only the last level of the first column is sorted. It is possible to modify the sorting of this column by clicking on its header.

Given for example the following list box, in which no specific sort is specified:

| Country          | Population |
|------------------|------------|
| France           |            |
| Brittany         |            |
| Rennes           | 200000     |
| Quimper          | 80000      |
| Brest            | 120000     |
| Lannion          | 20300      |
| Lorient          | 35000      |
| Normandy         |            |
| Caen             | 220000     |
| Deauville        | 4000       |
| Cherbourg        | 41000      |
| Auvergne         |            |
| Vichy            | 27000      |
| Moulins          | 20600      |
| Belgium          |            |
| Wallonia         |            |
| Namur            | 111000     |
| Liege            | 200000     |
| Flanders         |            |
| Antwerp          | 472000     |
| Louvain          | 95000      |
| Brussels-Capital |            |
| Brussels         | 155000     |

If you click on the "Population" header to sort the populations by ascending (or alternately descending) order, the data appear as follows:

As for all list boxes, you can [disable the standard sort mechanism](#) and manage sorts using programming.

## Selections and positions in hierarchical list boxes

A hierarchical list box displays a variable number of rows on screen according to the expanded/collapsed state of the hierarchical nodes. This does not however mean that the number of rows of the arrays vary. Only the display is modified, not the data. It is important to understand this principle because programmed management of hierarchical list boxes is always based on the data of the arrays, not on the displayed data. In particular, the break rows added automatically are not taken into account in the display options arrays (see below).

Let's look at the following arrays for example:

|        |          |         |
|--------|----------|---------|
| France | Brittany | Brest   |
| France | Brittany | Quimper |
| France | Brittany | Rennes  |

If these arrays are represented hierarchically, the row "Quimper" will not be displayed on the second row, but on the fourth, because of the two break rows that are added:

|          |
|----------|
| France   |
| Brittany |
| Brest    |
| Quimper  |
| Rennes   |

Regardless of how the data are displayed in the list box (hierarchically or not), if you want to change the row containing "Quimper" to bold, you must use the statement `Style{2} = bold`. Only the position of the row in the arrays is taken into account.

This principle is implemented for internal arrays that can be used to manage:

- colors
- background colors
- styles
- hidden rows
- selections

For example, if you want to select the row containing Rennes, you must pass:

```
->MyListbox{3}:=True
```

Non-hierarchical representation:

|        |          |         |
|--------|----------|---------|
| France | Brittany | Brest   |
| France | Brittany | Quimper |
| France | Brittany | Rennes  |

Hierarchical

representation:

|          |
|----------|
| France   |
| Brittany |
| Brest    |
| Quimper  |
| Rennes   |

If one or more rows are hidden because their parents are collapsed, they are no longer selected. Only the rows that are visible (either directly or by scrolling) can be selected. In other words, rows cannot be both hidden and selected.

As with selections, the `LISTBOX GET CELL POSITION` command will return the same values for a hierarchical list box and a non-hierarchical list box. This means that in both of the examples below, `LISTBOX GET CELL POSITION` will return the same position: (3;2).

Non-hierarchical representation:

|        |          |         |        |
|--------|----------|---------|--------|
| France | Brittany | Brest   | 120000 |
| France | Brittany | Quimper | 80000  |
| France | Brittany | Rennes  | 200000 |
| France | Normandy | Caen    | 75000  |

|          |        |
|----------|--------|
| France   |        |
| Brittany |        |
| Brest    | 120000 |
| Quimper  | 80000  |
| Rennes   | 200000 |
| Normandy |        |
| Caen     | 75000  |

Hierarchical representation:

When all the rows of a sub-hierarchy are hidden, the break line is automatically hidden. In the above example, if rows 1 to 3 are hidden, the "Brittany" break row will not appear.

## Break rows

If the user selects a break row, `LISTBOX GET CELL POSITION` returns the first occurrence of the row in the corresponding array. In the following case:

|          |        |
|----------|--------|
| France   |        |
| Brittany |        |
| Brest    | 120000 |
| Quimper  | 80000  |
| Rennes   | 200000 |
| Normandy |        |
| Caen     | 75000  |

... `LISTBOX GET CELL POSITION` returns (2;4). To select a break row by programming, you will need to use the `LISTBOX SELECT BREAK` command.

Break rows are not taken into account in the internal arrays used to manage the graphic appearance of list boxes (styles and colors). It is however possible to modify these characteristics for break rows via the graphic management commands for objects. You simply need to execute the appropriate commands on the arrays that constitute the hierarchy.

Given for example the following list box (the names of the associated arrays are specified in parentheses):

*Non-hierarchical representation:*

|           |        |
|-----------|--------|
| France    |        |
| Brittany  |        |
| Brest     | 120000 |
| Quimper   | 80000  |
| Rennes    | 200000 |
| Normandy  |        |
| Caen      | 75000  |
| Deauville | 4000   |

Hierarchical representation:

In hierarchical mode, break levels are not taken into account by the style modification arrays named `tStyle` and `tColors`. To modify the color or style of break levels, you must execute the following statements:

```
OBJECT SET RGB COLORS(T1;0x0000FF;0xB0B0B0)
OBJECT SET FONT STYLE(T2;Bold)
```

In this context, only the syntax using the array variable can function with the object property commands because the arrays do not have any associated object.

Result:

| France    |        |
|-----------|--------|
| Brittany  |        |
| Brest     | 120000 |
| Quimper   | 80000  |
| Rennes    | 200000 |
| Normandy  |        |
| Caen      | 75000  |
| Deauville | 4000   |

## Optimized management of expand/collapse

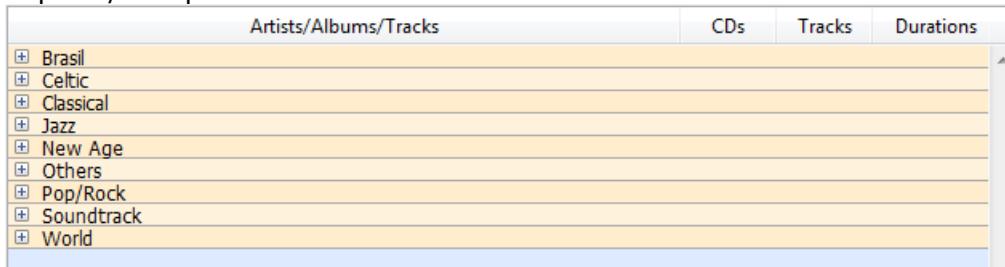
You can optimize hierarchical list boxes display and management using the `On Expand` and `On Collapse` form events.

A hierarchical list box is built from the contents of its arrays so it can only be displayed when all these arrays are loaded into memory. This makes it difficult to build large hierarchical list boxes based on arrays generated from data (through the `SELECTION TO ARRAY` command), not only because of the display speed but also the memory used.

Using the `On Expand` and `On Collapse` form events can overcome these constraints: for example, you can display only part of the hierarchy and load/unload the arrays on the fly, based on user actions. In the context of these events, the `LISTBOX GET CELL POSITION` command returns the cell where the user clicked in order to expand or collapse a row.

In this case, you must fill and empty arrays through the code. The principles to be implemented are:

- When the list box is displayed, only the first array must be filled. However, you must create a second array with empty values so that the list box displays the expand/collapse buttons:



- When a user clicks on an expand button, you can process the `On Expand` event. The `LISTBOX GET CELL POSITION` command returns the cell concerned and lets you build the appropriate hierarchy: you fill the first array with the repeated values and the second with the values sent from the `SELECTION TO ARRAY` command and you insert as many rows as needed in the list box using the `LISTBOX INSERT ROWS` command.
- When a user clicks on a collapse button, you can process the `On Collapse` event. The `LISTBOX GET CELL POSITION` command returns the cell concerned: you remove as many rows as needed from the list box using the `LISTBOX DELETE ROWS` command.

## Object arrays in columns

List box columns can handle object arrays. Since object arrays can contain different kinds of data, this powerful new feature allows you to mix different input types in the rows of a single column, and display various widgets as well. For example, you could insert a text input in the first row, a check box in the second, and a drop-down list in the third. Object arrays also provide access to new kinds of widgets, such as buttons or color pickers.

The following list box was designed using an object array:

| Label                        | Value                                     |
|------------------------------|-------------------------------------------|
| Document Name                | MyReport                                  |
| Document Type                | PDF                                       |
| Reference                    | 123456                                    |
| Category                     | <input type="button" value="..."/>        |
| Include Abstract             | <input checked="" type="checkbox"/>       |
| Printable area size (height) | 297 <input type="button" value="mm"/>     |
| Printable area size (width)  | 210 <input type="button" value="mm"/>     |
| Show Preview                 | <input type="button" value="Preview..."/> |

## Configuring an object array column

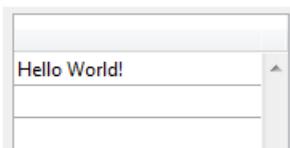
To assign an object array to a list box column, you just need to set the object array name in either the Property list ("Variable Name" field), or using the [LISTBOX INSERT COLUMN](#) command, like with any array-based column. In the Property list, you can now select Object as a "Expression Type" for the column:

Standard properties related to coordinates, size, and style are available for object columns. You can define them using the Property list, or by programming the style, font color, background color and visibility for each row of an object-type list box column. These types of columns can also be hidden.

However, the Data Source theme is not available for object-type list box columns. In fact, the contents of each column cell are based on attributes found in the corresponding element of the object array. Each array element can define:

the value type (mandatory): text, color, event, etc. the value itself (optional): used for input/output. the cell content display (optional): button, list, etc. additional settings (optional): depend on the value type To define these properties, you need to set the appropriate attributes in the object (available attributes are listed below). For example, you can write "Hello World!" in an object column using this simple code:

```
ARRAY OBJECT(obColumn;0) //column array
C_OBJECT($ob) //first element
OB SET($ob;"valueType";"text") //defines the value type (mandatory)
OB SET($ob;"value";"Hello World!") //defines the value
APPEND TO ARRAY(obColumn;$ob)
```



Display format and entry filters cannot be set for an object column. They automatically depend on the value type.

## valueType and data display

When a list box column is associated with an object array, the way a cell is displayed, entered, or edited, is based on the valueType attribute of the array element.

Supported valueType values are:

- "text": for a text value
- "real": for a numeric value that can include separators like a \<space>, <. >, or < , >
- "integer": for an integer value
- "boolean": for a True/False value
- "color": to define a background color
- "event": to display a button with a label.

4D uses default widgets with regards to the "valueType" value (i.e., a "text" is displayed as a text input widget, a "boolean" as a check box), but alternate displays are also available through options (e.g., a real can also be represented as a drop-down menu). The following table shows the default display as well as alternatives for each type of value:

| <b>valueType</b> | <b>Default widget</b>                          | <b>Alternative widget(s)</b>                                                                   |
|------------------|------------------------------------------------|------------------------------------------------------------------------------------------------|
| text             | text input                                     | drop-down menu (required list) or combo box (choice list)                                      |
| real             | controlled text input (numbers and separators) | drop-down menu (required list) or combo box (choice list)                                      |
| integer          | controlled text input (numbers only)           | drop-down menu (required list) or combo box (choice list) or three-states check box            |
| boolean          | check box                                      | drop-down menu (required list)                                                                 |
| color            | background color                               | text                                                                                           |
| event            | button with label                              | All widgets can have an additional unit toggle button or ellipsis button attached to the cell. |

You set the cell display and options using specific attributes in each object (see below).

## Display formats and entry filters

You cannot set display formats or entry filters for columns of object-type list boxes. They are automatically defined according to the value type. These are listed in the following table:

| <b>Value type</b> | <b>Default format</b>                                      | <b>Entry control</b>                                        |
|-------------------|------------------------------------------------------------|-------------------------------------------------------------|
| text              | same as defined in object                                  | any (no control)                                            |
| real              | same as defined in object (using system decimal separator) | "0-9" and "." and "-"                                       |
| integer           | same as defined in object                                  | "0-9" and "." if min>=0<br>"0-9" and "-"<br>"0-9" if min>=0 |
| Boolean           | check box                                                  | N/A                                                         |
| color             | N/A                                                        | N/A                                                         |
| event             | N/A                                                        | N/A                                                         |

## Attributes

Each element of the object array is an object that can contain one or more attributes that will define the cell contents and data display (see example above).

The only mandatory attribute is "valueType" and its supported values are "text", "real", "integer", "boolean", "color", and "event". The following table lists all the attributes supported in list box object arrays, depending on the "valueType" value (any other attributes are ignored). Display formats are detailed and examples are provided below.

|                       | <b>valueType</b>                        | <b>text</b> | <b>real</b> | <b>integer</b> | <b>boolean</b> | <b>color</b> | <b>event</b> |
|-----------------------|-----------------------------------------|-------------|-------------|----------------|----------------|--------------|--------------|
| <i>Attributes</i>     | <i>Description</i>                      |             |             |                |                |              |              |
| value                 | cell value (input or output)            | x           |             | x              |                |              |              |
| min                   | minimum value                           |             | x           | x              |                |              |              |
| max                   | maximum value                           |             | x           | x              |                |              |              |
| behavior              | "threeStates" value                     |             |             | x              |                |              |              |
| requiredList          | drop-down list defined in object        | x           | x           | x              |                |              |              |
| choiceList            | combo box defined in object             | x           | x           | x              |                |              |              |
| requiredListReference | 4D list ref, depends on "saveAs" value  | x           | x           | x              |                |              |              |
| requiredListName      | 4D list name, depends on "saveAs" value | x           | x           | x              |                |              |              |
| saveAs                | "reference" or "value"                  | x           | x           | x              |                |              |              |
| choiceListReference   | 4D list ref, display combo box          | x           | x           | x              |                |              |              |
| choiceListName        | 4D list name, display combo box         | x           | x           | x              |                |              |              |
| unitList              | array of X elements                     | x           | x           | x              |                |              |              |
| unitReference         | index of selected element               | x           | x           | x              |                |              |              |
| unitsListReference    | 4D list ref for units                   | x           | x           | x              |                |              |              |
| unitsListName         | 4D list name for units                  | x           | x           | x              |                |              |              |
| alternateButton       | add an alternate button                 | x           | x           | x              |                | x            | x            |

## value

Cell values are stored in the "value" attribute. This attribute is used for input as well as output. It can also be used to define default values when using lists (see below).

```

ARRAY OBJECT(obColumn;0) //column array
C_OBJECT($ob1)
$entry:="Hello world!"
OB SET($ob1;"valueType";"text")
OB SET($ob1;"value";$entry) // if the user enters a new value, $entry
will contain the edited value
C_OBJECT($ob2)
OB SET($ob2;"valueType";"real")
OB SET($ob2;"value";2/3)
C_OBJECT($ob3)
OB SET($ob3;"valueType";"boolean")
OB SET($ob3;"value";True)

APPEND TO ARRAY(obColumn;$ob1)
APPEND TO ARRAY(obColumn;$ob2)
APPEND TO ARRAY(obColumn;$ob3)

```

| Header1                             |       |
|-------------------------------------|-------|
| Hello                               | world |
| 0,6666666666667                     |       |
| <input checked="" type="checkbox"/> |       |

Null values are supported and result in an empty cell.

## min and max

When the "valueType" is "real" or "integer", the object also accepts min and max attributes with appropriate values (values must be of the same type as the valueType).

These attributes can be used to control the range of input values. When a cell is validated (when it loses the focus), if the input value is lower than the min value or greater than the max value, then it is rejected. In this case, the previous value is maintained and a tip displays an explanation.

```
C_OBJECT($ob3)
$entry3:=2015
OB SET($ob3;"valueType";"integer")
OB SET($ob3;"value";$entry3)
OB SET($ob3;"min";2000)
OB SET($ob3;"max";3000)
```

## behavior

The behavior attribute provides variations to the regular representation of values. In 4D v15, a single variation is proposed:

| Attribute | Available value(s) | valueType(s) | Description                                                                                                                                                                           |
|-----------|--------------------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| behavior  | threeStatesinteger |              | Represents a numeric value as a three-states check box.<br>2=semi-checked, 1=checked, 0=unchecked, -1=invisible, -2=unchecked disabled, -3=checked disabled, -4=semi-checked disabled |

```
C_OBJECT($ob3)
OB SET($ob3;"valueType";"integer")

OB SET($ob3;"value";-3)
C_OBJECT($ob4)
OB SET($ob4;"valueType";"integer")
OB SET($ob4;"value";-3)
OB SET($ob4;"behavior";"threeStates")
```

## requiredList and choiceList

When a "choiceList" or a "requiredList" attribute is present inside the object, the text input is replaced by a drop-down list or a combo box, depending of the attribute:

- If the attribute is "choiceList", the cell is displayed as a combo box. This means that the user can select or type a value.
- If the attribute is "requiredList" then the cell is displayed as a drop-down list and the user can only select one of the values provided in the list.

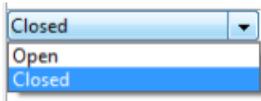
In both cases, a "value" attribute can be used to preselect a value in the widget.

The widget values are defined through an array. If you want to assign an existing 4D list to the widget, you need to use the "requiredListReference", "requiredListName", "choiceListReference", or "choiceListName" attributes.

Examples:

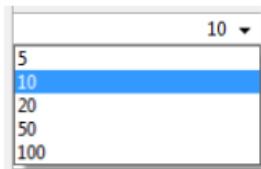
- You want to display a drop-down list with only two options: "Open" or "Closed". "Closed" must be preselected:

```
ARRAY TEXT ($RequiredList;0)
APPEND TO ARRAY ($RequiredList;"Open")
APPEND TO ARRAY ($RequiredList;"Closed")
C_OBJECT ($ob)
OB SET ($ob;"valueType";"text")
OB SET ($ob;"value";"Closed")
OB SET ARRAY ($ob;"requiredList";$RequiredList)
```



- You want to accept any integer value, but display a combo box to suggest the most common values:

```
ARRAY LONGINT ($ChoiceList;0)
APPEND TO ARRAY ($ChoiceList;5)
APPEND TO ARRAY ($ChoiceList;10)
APPEND TO ARRAY ($ChoiceList;20)
APPEND TO ARRAY ($ChoiceList;50)
APPEND TO ARRAY ($ChoiceList;100)
C_OBJECT ($ob)
OB SET ($ob;"valueType";"integer")
OB SET ($ob;"value";10) //10 as default value
OB SET ARRAY ($ob;"choiceList";$ChoiceList)
```



## **requiredListName and requiredListReference**

The "requiredListName" and "requiredListReference" attributes allow you to use, in a list box cell, a list defined in 4D either in Design mode (in the Lists editor of the Tool box) or by programming (using the New list command). The cell will then be displayed as a drop-down list. This means that the user can only select one of the values provided in the list.

Use "requiredListName" or "requiredListReference" depending on the origin of the list: if the list comes from the Tool box, you pass a name; otherwise, if the list has been defined by programming, you pass a reference. In both cases, a "value" attribute can be used to preselect a value in the widget.

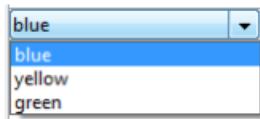
- If you want to define these values through a simple array, you need to use the "requiredList" attribute.
- If the list contains text items representing real values, the decimal separator must be a period ("."), regardless of the local settings, e.g.: "17.6" "1234.456".

Examples:

- You want to display a drop-down list based on a "colors" list defined in the Tool box (containing the values "blue", "yellow", and "green"), save it as a value and display "blue" by default:



```
C_OBJECT($ob)
OB SET($ob;"valueType";"text")
OB SET($ob;"saveAs";"value")
OB SET($ob;"value";"blue")
OB SET($ob;"requiredListName";"colors")
```



- You want to display a drop-down list based on a list defined by programming and save it as a reference:

```
<>List:=New list
APPEND TO LIST(<>List;"Paris";1)
APPEND TO LIST(<>List;"London";2)
APPEND TO LIST(<>List;"Berlin";3)
APPEND TO LIST(<>List;"Madrid";4)
C_OBJECT($ob)
OB SET($ob;"valueType";"integer")
OB SET($ob;"saveAs";"reference")
OB SET($ob;"value";2) //displays London by default
OB SET($ob;"requiredListReference";<>List)

```

### **choiceListName and choiceListReference**

The "choiceListName" and "choiceListReference" attributes allow you to use, in a list box cell, a list defined in 4D either in Design mode (in the Tool box) or by programming (using the New list command). The cell is then displayed as a combo box, which means that the user can select or type a value.

Use "choiceListName" or "choiceListReference" depending on the origin of the list: if the list comes from the Tool box, you pass a name; otherwise, if the list has been defined by programming, you pass a reference. In both cases, a "value" attribute can be used to preselect a value in the widget.

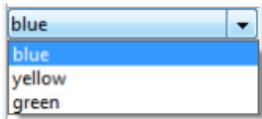
- If you want to define these values through a simple array, you need to use the "choiceList" attribute.
- If the list contains text items representing real values, the decimal separator must be a period ("."), regardless of the local settings, e.g.: "17.6" "1234.456".

## Example:

You want to display a combo box based on a "colors" list defined in the Tool box (containing the values "blue", "yellow", and "green") and display "green" by default:



```
C_OBJECT($ob)
OB SET($ob;"valueType";"text")
OB SET($ob;"value";"blue")
OB SET($ob;"choiceListName";"colors")
```



## **unitsList, unitsListName, unitsListReference and unitReference**

You can use specific attributes to add units associated with cell values (e.g.: "10 cm", "20 pixels", etc.). To define the unit list, you can use one of the following attributes:

- "unitsList": an array containing the x elements used to define the available units (e.g.: "cm", "inches", "km", "miles", etc.). Use this attribute to define units within the object.
- "unitsListReference": a reference to a 4D list containing available units. Use this attribute to define units with a 4D list created with the [New list](#) command.
- "unitsListName": a name of a design-based 4D list that contains available units. Use this attribute to define units with a 4D list created in the Tool box.

Regardless of the way the unit list is defined, it can be associated with the following attribute:

- "unitReference": a single value that contains the index (from 1 to x) of the selected item in the "unitList", "unitsListReference" or "unitsListName" values list.

The current unit is displayed as a button that cycles through the "unitList", "unitsListReference" or "unitsListName" values each time it is clicked (e.g., "pixels" -> "rows" -> "cm" -> "pixels" -> etc.)

## Example:

We want to set up a numeric input followed by two possible units: "rows" or "pixels". The current value is "2" + "lines". We use values defined directly in the object ("unitsList" attribute):

```
ARRAY TEXT($_units;0)
APPEND TO ARRAY($_units;"lines")
APPEND TO ARRAY($_units;"pixels")
C_OBJECT($ob)
OB SET($ob;"valueType";"integer")
OB SET($ob;"value";2) // 2 "units"
OB SET($ob;"unitReference";1) //"lines"
OB SET ARRAY($ob;"unitsList";$_units)
```



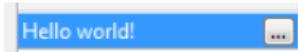
## **alternateButton**

If you want to add an ellipsis button [...] to a cell, you just need to pass the "alternateButton" with the True value in the object. The button will be displayed in the cell automatically.

When this button is clicked by a user, an `On Alternate Click` event will be generated, and you will be able to handle it however you want (see the "Event management" paragraph for more information).

Example:

```
C_OBJECT($ob1)
$entry:="Hello world!"
OB SET($ob;"valueType";"text")
OB SET($ob;"alternateButton";True)
OB SET($ob;"value";$entry)
```



## **color valueType**

The "color" valueType allows you to display either a color or a text.

- If the value is a number, a colored rectangle is drawn inside the cell. Example:

```
``` `4d
C_OBJECT($ob4)
OB SET($ob4;"valueType";"color")
OB SET($ob4;"value";0x00FF0000)
```
```



- If the value is a text, then the text is displayed (e.g.: "value";"Automatic").

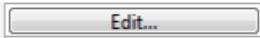
## **event valueType**

The "event" valueType displays a simple button that generates an `On Clicked` event when clicked. No data or value can be passed or returned.

Optionally, you can pass a "label" attribute.

Example:

```
C_OBJECT($ob)
OB SET($ob;"valueType";"event")
OB SET($ob;"label";"Edit...")
```



## **Event management**

Several events can be handled while using an object list box array:

- **On Data Change:** An `On Data Change` event is triggered when any value has been modified either:
  - in a text input zone
  - in a drop-down list

- in a combo box area
- in a unit button (switch from value x to value x+1)
- in a check box (switch between checked/unchecked)
- **On Clicked:** When the user clicks on a button installed using the "event" *valueType* attribute, an `On Clicked` event will be generated. This event is managed by the programmer.
- **On Alternative Click:** When the user clicks on an ellipsis button ("alternateButton" attribute), an `On Alternative Click` event will be generated. This event is managed by the programmer.

## Picture Button

A picture button is similar to a [standard button](#). However unlike a standard button (which accepts three states: enabled, disabled and clicked), a picture button has a different image to represent each state.

Picture buttons can be used in two ways:

- As command buttons in a form. In this case, the picture button generally includes four different states: enabled, disabled, clicked and rolled over.  
For example, a table of thumbnails that has one row of four columns, each thumbnail corresponds to the Default, Clicked, Roll over, and Disabled states.

Property	JSON name	Value
Rows	rowCount	1
Columns	columnCount	4
Switch back when Released	switchBackWhenReleased	true
Switch when Roll Over	switchWhenRollover	true
Use Last Frame as Disabled	useLastFrameAsDisabled	true

- As a picture button letting the user choose among several choices. In this case, a picture button can be used in place of a pop-up picture menu. With [Picture Pop-up Menus](#), all choices are displayed simultaneously (as the items in the pop-up menu), while the picture button displays the choices consecutively (as the user clicks the button).

Here is an example of a picture button. Suppose you want to give the users of a custom application the opportunity to choose the interface language for the application. You implement the option as a picture button in a custom properties dialog box:



Clicking the object changes the picture.

## Using picture buttons

You can implement a picture button in the following manner:

- First, prepare a single graphic in which the series of images are arranged in a row, a column, or a row-by-column grid.



You can organize pictures as columns, rows, or a row-by-column grid (as shown above). When organizing pictures as a grid, they are numbered from left to right, row

by row, beginning with 0. For example, the second picture of the second row of a grid that consists of two rows and three columns, is numbered 4 (The UK flag in the example above).

2. Next, make sure the image is in your project's Resources and enter the path in the [Pathname](#) property.
3. Define the graphic's [Rows and Columns](#) properties.
4. Specify when the images change by selecting appropriate [animation](#) properties.

## Animation

In addition to the standard positioning and appearance settings, you can set some specific properties for picture buttons, especially concerning how and when the pictures are displayed. These property options can be combined to enhance your picture buttons.

- By default (when no [animation option](#) is selected), a picture button displays the next picture in the series when the user clicks; it displays the previous picture in the series when the user holds down the **Shift** key and clicks. When the user reaches the last picture in the series, the picture does not change when the user clicks again. In other words, it does not cycle back to the first picture in the series.

The following other modes are available:

- [Loop back to first frame](#)
- [Switch back when Released](#)
- [Switch when Roll Over](#)
- [Switch continuously on clicks](#)
- [Use Last Frame as Disabled](#)
- [Use Last frame as disabled](#)

The [associated variable](#) of the picture button returns the index number, in the thumbnail table, of the current picture displayed. The numbering of pictures in the table begins with 0.

## Supported Properties

[Bold](#) - [Border Line Style](#) - [Bottom](#) - [Button Style](#) - [Class](#) - [Columns](#) - [Focusable](#) - [Font](#) - [Font Color](#) - [Height](#) - [Help Tip](#) - [Horizontal Sizing](#) - [Italic](#) - [Left](#) - [Loop back to first frame](#) - [Object Name](#) - [Pathname](#) - [Right](#) - [Rows](#) - [Shortcut](#) - [Standard action](#) - [Switch back when released](#) - [Switch continuously on clicks](#) - [Switch every x ticks](#) - [Title](#) - [Switch when roll over](#) - [Top](#) - [Type](#) - [Use Last frame as disabled](#) - [Variable or Expression](#) - [Vertical Sizing](#) - [Visibility](#) - [Width](#)

## Picture Pop-up Menu

A picture pop-up menu is a pop-up menu that displays a two-dimensional array of pictures. A picture pop-up menu can be used instead of a [picture button](#). The creation of the picture to use with a picture pop-up menu is similar to the creation of a picture for a picture button. The concept is the same as for [button grids](#), except that the graphic is used as a pop-up menu instead of a form object.

## Using picture pop-up menus

To create a picture pop-up menu, you need to [refer to a picture](#). The following example allows you to select the interface language by selecting it from a picture pop-up menu. Each language is represented by the corresponding flag:



## Programming

You can manage picture pop-up menus using methods. As with [button grids](#), variables associated with picture pop-up menus are set to the value of the selected element in the picture pop-up menu. If no element is selected, the value is 0. Elements are numbered, row by row, from left to right starting with the top row.

### Goto page

You can assign the `gotoPage` [standard action](#) to a picture pop-up menu. When that action is selected, 4D will automatically display the page of the form that corresponds to the position of the picture selected in the picture array. Elements are numbered from left to right and top to bottom, beginning with the top left corner.

For example, if the user selects the 3rd element, 4D will display the third page of the current form (if it exists). If you want to manage the effect of a click yourself, select `No action`.

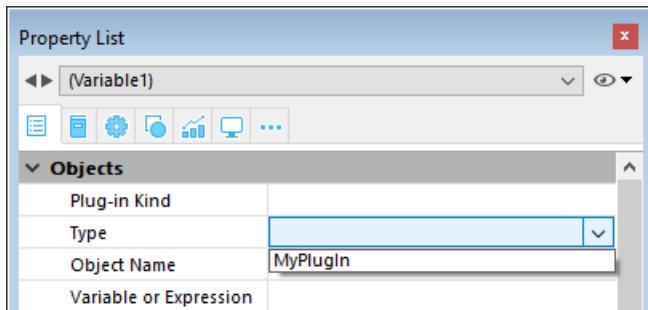
## Supported Properties

[Bold](#) - [Border Line Style](#) - [Bottom](#) - [Class](#) - [Columns](#) - [Height](#) - [Help Tip](#) - [Horizontal Sizing](#) - [Left](#) - [Object Name](#) - [Pathname](#) - [Right](#) - [Rows](#)- [Standard action](#) - [Top](#) - [Type](#) - [Variable or Expression](#) - [Vertical Sizing](#) - [Visibility](#) - [Width](#)

## Plug-in Area

A plug-in area is an area on the form that is completely controlled by a plug-in. The ability to incorporate plug-ins into forms gives you unlimited possibilities when creating custom applications. A plug-in can perform a simple task such as displaying a digital clock on a form, or a complex task such as providing full-featured word processing, spreadsheet, or graphics capabilities.

When opening an application, 4D creates an internal list of the plug-ins [installed in your application](#). Once you have inserted a Plug-in Area in a form, you can assign a plug-in to the area directly in the **Type** list in the Property List:



Some plug-ins, such as 4D Internet Commands, cannot be used in forms or external windows. When a plug-in cannot be used in a form, it does not appear in the plug-in list of the Property List.

If you draw a plug-in area that is too small, 4D will display it as a button whose title is the name of the variable associated with the area. During execution, the user can click on this button in order to open a specific window displaying the plug-in.

## Advanced properties

If advanced options are provided by the author of the plug-in, a **Plug-in** theme containing an [Advanced Properties](#) button may be enabled in the Property list. In this case, you can click this button to set these options, usually through a custom dialog box.

## Installing plug-ins

To add a plug-in in your 4D environment, you first need to quit 4D. Plug-ins are loaded when you launch 4D. For more information about the installation of plug-ins, refer to [Installing plugins or components](#).

## Creating plug-ins

If you are interested in designing your own plug-ins, you can receive extensive information about writing and implementing plug-ins. 4D provides a [complete kit \(on github\)](#) to help you write custom plug-ins.

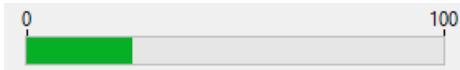
## Supported Properties

[Border Line Style](#) - [Bottom](#) - [Advanced Properties](#) - [Class](#) - [Draggable](#) - [Droppable](#) - [Expression Type](#) - [Focusable](#) - [Height](#) - [Horizontal Sizing](#) - [Left](#) - [Method](#) - [Object Name](#) - [Plug-in Kind](#) - [Right](#) - [Top](#) - [Type](#) - [Variable or Expression](#) - [Vertical Sizing](#) - [Visibility](#) - [Width](#)



## Progress Indicator

A progress indicator (also called "thermometer") is designed to display or set numeric or date/time values graphically.



## Using indicators

You can use indicators either to display or set values. For example, if a progress indicator is given a value by a method, it displays the value. If the user drags the indicator point, the value changes. The value can be used in another object such as a field or an enterable or non-enterable object.

The variable associated with the indicator controls the display. You place values into, or use values from, the indicator using methods. For example, a method for a field or enterable object could be used to control a progress indicator:

```
vTherm:=[Employees]Salary
```

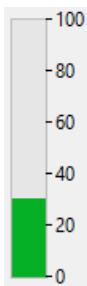
This method assigns the value of the Salary field to the vTherm variable. This method would be attached to the Salary field.

Conversely, you could use the indicator to control the value in a field. The user drags the indicator to set the value. In this case the method becomes:

```
[Employees]Salary:=vTherm
```

The method assigns the value of the indicator to the Salary field. As the user drags the indicator, the value in the Salary field changes.

## Default thermometer



The thermometer is the basic progress indicator.

You can display horizontal or vertical thermometers bars. This is determined by the shape of the object that you draw.

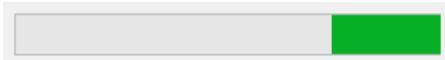
Multiple graphical options are available: minimum/maximum values, graduations, steps.

## Supported Properties

[Barber shop](#) - [Bold](#) - [Border Line Style](#) - [Bottom](#) - [Class](#) - [Display graduation](#) - [Enterable](#) - [Execute object method](#) - [Expression Type](#) (only "integer", "number", "date", or "time") - [Font](#) - [Font Color](#) - [Font Size](#) - [Height](#) - [Italic](#) - [Graduation step](#) - [Help Tip](#) - [Horizontal Sizing](#) - [Label Location](#) - [Left](#) - [Maximum](#) - [Minimum](#) - [Number Format](#) -

[Object Name](#) - [Right](#) - [Step](#) - [Top](#) - [Type](#) - [Underline](#) - [Variable or Expression](#) - [Vertical Sizing](#) - [Visibility](#) - [Width](#)

## Barber shop



**Barber shop** is a variant of the default thermometer. To enable this variant, you need to set the [Barber shop](#) property.

In JSON code, just remove "max" property from a default thermometer object to enable the Barber shop variant.

Barber shop displays a continuous animation, like the [spinner](#). These thermometers are generally used to indicate to the user that the program is in the process of carrying out a long operation. When this thermometer variant is selected, [graphical Scale properties](#) are not available.

When the form is executed, the object is not animated. You manage the animation by passing a value to its [associated variable or expression](#):

- 1 = Start animation,
- 0 = Stop animation.

### Supported Properties

[Barber shop](#) - [Bold](#) - [Border Line Style](#) - [Bottom](#) - [Class](#) - [Enterable](#) - [Execute object method](#) - [Expression Type](#) (only "integer", "number", "date", or "time") - [Font](#) - [Font Color](#) - [Font Size](#) - [Height](#) - [Help Tip](#) - [Horizontal Sizing](#) - [Italic](#) - [Left](#) - [Object Name](#) - [Right](#) - [Top](#) - [Type](#) - [Underline](#) - [Variable or Expression](#) - [Vertical Sizing](#) - [Visibility](#) - [Width](#)

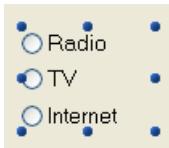
### See also

- [rulers](#)
- [steppers](#)

## Radio Button

Radio buttons are objects that allow the user to select one of a group of buttons.

Usually, a radio button shows a small bullseye with text. However, radio buttons can have [various appearances](#).



A radio button is selected:

- when the user clicks on it
- when it has the focus and the user presses the **Space bar** key.

## Configuring radio buttons

Radio buttons are used in coordinated sets: only one button at a time can be selected in the set. In order to operate in a coordinated manner, a set of radio buttons must share the same [Radio Group](#) property.

Radio buttons are controlled with methods. Like all buttons, a radio button is set to 0 when the form is first opened. A method associated with a radio button executes when the button is selected. The following is an example of a group of radio buttons used in a video collection database to enter the speed of the recording (SP, LP, or EP):



Selecting one radio button in a group sets that button to 1 and all of the others in the group to 0. Only one radio button can be selected at a time.

You can associate [Boolean type expressions](#) with radio buttons. In this case, when a radio button in a group is selected, its variable is True and the variables for the group's other radio buttons are False.

The value contained in a radio button object is not saved automatically (except if it is the representation of a Boolean field); radio button values must be stored in their variables and managed with methods.

## Button Styles

Radio [button styles](#) control radio button's general appearance as well as its available properties. It is possible to apply different predefined styles to radio buttons. However, the same button style must be applied to all radio buttons in a group so that they work as expected.

4D provides radio buttons in the following predefined styles:

### Regular

The Regular radio button style is a standard system button (*i.e.*, a small bullseye with

text) which executes code when a user clicks on it.



In addition to initiating code execution, the Regular radio button style changes bullseye color when being hovered.

## Flat

The Flat radio button style is a standard system button (*i.e.*, a small bullseye with text) which executes code when a user clicks on it.



By default, the Flat style has a minimalist appearance. The Flat button style's graphic nature is particularly useful for forms that will be printed.

## Toolbar

The Toolbar radio button style is primarily intended for integration in a toolbar.

By default, the Toolbar style has a transparent background with a label in the center. The appearance of the button can be different when the cursor hovers over it depending on the OS:

- *Windows* - the button is highlighted.



- *macOS* - the highlight of the button never appears.

## Bevel

The Bevel radio button style is similar to the [Toolbar](#) style's behavior, except that it has a light gray background and a gray outline. The appearance of the button can be different when the cursor hovers over it depending on the OS:

- *Windows* - the button is highlighted.



- *macOS* - the highlight of the button never appears.

## Rounded Bevel

The Rounded Bevel button style is nearly identical to the [Bevel](#) style except, depending on the OS, the corners of the button may be rounded.

- *Windows* - the button is identical to the [Bevel](#) style.



- *macOS* - the corners of the button are rounded.

## OS X Gradient

The OS X Gradient button style is nearly identical to the [Bevel](#) style except, depending on the OS, it may have a two-toned appearance.

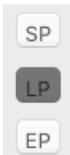
- *Windows* - the button is identical to the [Bevel](#) style.
- *macOS* - the button is displayed as a two-tone system button.

## OS X Textured

The OS X Textured radio button style is nearly identical to the [Toolbar](#) style except, depending on the OS, it may have a different appearance and does not display hover.

By default, the OS X Textured style appears as:

- *Windows* - a toolbar-like button with a label in the center and the background is always displayed.
- *macOS* - a standard system button displaying a color change from light to dark gray. Its height is predefined: it is not possible to enlarge or reduce it.



## Office XP

The Office XP button style combines the appearance of the [Regular](#) style (standard system button) with the [Toolbar](#) style's behavior.

The colors (highlight and background) of a button with the Office XP style are based on the system colors. The appearance of the button can be different when the cursor hovers over it depending on the OS:

- *Windows* - its background only appears when the mouse rolls over it.



- *macOS* - its background is always displayed.

## Collapse/Expand

This button style can be used to add a standard collapse/expand icon. These buttons are used natively in hierarchical lists. In Windows, the button looks like a [+] or a [-];

in macOS, it looks like a triangle pointing right or down.



### (!) INFO

The Collapse/Expand style is named "disclosure" in the [button style JSON Grammar](#).

## Disclosure

The disclosure radio button style displays the radio button as a standard disclosure button, usually used to show/hide additional information. The button symbol points downwards with value 0 and upwards with value 1.



### (!) INFO

The Disclosure style is named "roundedDisclosure" in the [button style JSON Grammar](#).

## Custom

The Custom radio button style accepts a personalized background picture and allows managing additional parameters such as [icon offset](#) and [margins](#).

## Supported properties

All radio buttons share the same set of basic properties:

[Bold](#) - [Bottom](#) - [Button Style](#) - [Class](#) - [Expression Type](#) - [Focusable](#) - [Font](#) - [Font Color](#) - [Height](#) - [Help Tip](#) - [Horizontal Alignment\(1\)](#) - [Horizontal Sizing](#) - [Image hugs title\(2\)](#) - [Italic](#) - [Left](#) - [Number of States\(2\)](#) - [Method](#) - [Object Name](#) - [Radio Group](#) - [Picture pathname\(2\)](#) - [Right](#) - [Save value](#) - [Shortcut](#) - [Title](#) - [Title/Picture Position\(2\)](#) - [Top](#) - [Type](#) - [Underline](#) - [Variable or Expression](#) - [Vertical Sizing](#) - [Visibility](#) - [Width](#)

(1) Not supported by the [Regular](#) and [Flat](#) styles.

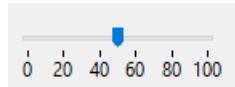
(2) Not supported by the [Regular](#), [Flat](#), [Disclosure](#) and [Collapse/Expand](#) styles.

Additional specific properties are available depending on the [button style](#):

- Custom: [Background pathname](#) - [Horizontal Margin](#) - [Icon Offset](#) - [Vertical Margin](#)

## Ruler

The ruler is a standard interface object used to set or get values using a cursor moved along its graduations.



You can assign its [associated variable or expression](#) to an enterable area (field or variable) to store or modify the current value of the object.

For more information, please refer to [Using indicators](#) in the "Progress Indicator" page.

### Supported Properties

[Bold](#) - [Border Line Style](#) - [Bottom](#) - [Class](#) - [Display graduation](#) - [Enterable](#) - [Execute object method](#) - [Expression Type](#) - [Height](#) - [Graduation step](#) - [Help Tip](#) - [Horizontal Sizing](#) - [Label Location](#) - [Left](#) - [Maximum](#) - [Minimum](#) - [Number Format](#) - [Object Name](#) - [Right](#) - [Step](#) - [Top](#) - [Type](#) - [Variable or Expression](#) - [Vertical Sizing](#) - [Visibility](#) - [Width](#)

### See also

- [progress indicators](#)
- [steppers](#)

# Shapes

Shapes are [static objects](#) that can be added to 4D forms.

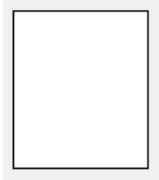
4D forms support the following basic shapes:

- rectangles
- lines
- ovals

## Rectangle

A static rectangle is a decorative object for forms. Rectangles are constrained to squared shapes.

The design of rectangles is controlled through many properties (color, line thickness, pattern, etc.). Specifically, the [roundness](#) of its corners can be defined.



### JSON Example:

```
"myRectangle": {  
    "type": "rectangle",      //define the type of object  
    "left": 60,                //left position on the form  
    "top": 160,               //top position on the form  
    "width": 100,              //width of the object  
    "height": 20,              //height of the object  
    "borderRadius": 20        //define the roundness of the corners  
}
```

### Supported Properties

[Bottom](#) - [Class](#) - [Corner radius](#) - [Dotted Line Type](#) - [Fill Color](#) - [Height](#) - [Horizontal Sizing](#) - [Left](#) - [Line Color](#) - [Line Width](#) - [Object Name](#) - [Right](#) - [Top](#) - [Type](#) - [Vertical Sizing](#) - [Visibility](#) - [Width](#)

## Line

A static line is a decorative object for forms, drawn between two plots. Lines can be horizontal, vertical, or of any angle shapes.

The design of lines is controlled through many properties (color, line thickness, etc.).

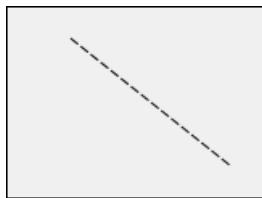
### startPoint property

The `startPoint` JSON property defines from which coordinate to draw the line (see example).

the `startPoint` property is not exposed in the Property List, where the line drawing direction is visible.

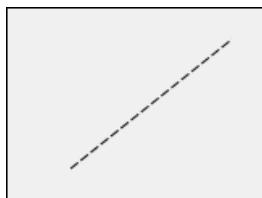
## JSON Examples:

```
"myLine": {  
    "type": "line",  
    "left": 20,  
    "top": 40,  
    "width": 100,  
    "height": 80,  
    "startPoint": "topLeft", //first direction  
    "strokeDashArray": "6 2" //dashed  
}
```



Result:

```
"myLine": {  
    "type": "line",  
    "left": 20,  
    "top": 40,  
    "width": 100,  
    "height": 80,  
    "startPoint": "bottomLeft", //2nd direction  
    "strokeDashArray": "6 2" //dashed  
}
```



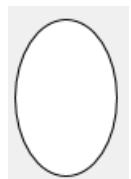
Result:

## Supported Properties

[Bottom](#) - [Class](#) - [Dotted Line Type](#) - [Height](#) - [Horizontal Sizing](#) - [Left](#) - [Line Color](#) - [Line Width](#) - [Object Name](#) - [Right](#) - [startPoint](#) - [Top](#) - [Type](#) - [Vertical Sizing](#) - [Visibility](#) - [Width](#)

## Oval

A static oval is a decorative object for forms. Oval objects can be used to draw circular shapes (when [width](#) and [height](#) properties are equal).



## JSON Example:

```
"myOval": {  
    "type": "oval",           //define the type of object  
    "left": 60,              //left position on the form  
    "top": 160,              //top position on the form  
    "width": 100,             //width of the object  
    "height": 20,             //height of the object  
    "fill": "blue"           //define the background color  
}
```

## Supported Properties

[Bottom](#) - [Class](#) - [Dotted Line Type](#) - [Fill Color](#) - [Height](#) - [Horizontal Sizing](#) - [Left](#) - [Line Color](#) - [Line Width](#) - [Object Name](#) - [Right](#) - [Top](#) - [Type](#) - [Vertical Sizing](#) - [Visibility](#) - [Width](#)

## Spinner

The spinner is a circular indicator that displays a continuous animation, like the [Barber shop](#).

You use this type of object to indicate that an operation such as establishing a network connection or performing a calculation is underway. When this indicator is selected, [graphical Scale properties](#) are not available.

When the form is executed, the object is not animated. You manage the animation by passing a value to its [associated variable or expression](#):

- 1 (or any value other than 0) = Start animation,
- 0 = Stop animation

### Supported Properties

[Border Line Style](#) - [Bottom](#) - [Class](#) - [Expression Type](#) - [Height](#) - [Help Tip](#) - [Horizontal Sizing](#) - [Left](#) - [Object Name](#) - [Right](#) - [Top](#) - [Type](#) - [Variable or Expression](#) - [Vertical Sizing](#) - [Visibility](#) - [Width](#)

## Splitter

A splitter divides a form into two areas, allowing the user to enlarge and reduce the areas by moving the splitter one way or the other. A splitter can be either horizontal or vertical. The splitter takes into account each object's resizing properties, which means that you can completely customize your application's interface. A splitter may or may not be a "pusher."

Splitter are used for example in output forms so that columns can be resized:

Job Title:	Company:
Secretary	Howard Battery Co.
Salesperson	Howard Battery Co.
Salesperson	Howard Battery Co.
Supervisor	Howard Battery Co.
Director	BluePines

Some of the splitter's general characteristics:

- You can place as many splitters as you want in any type of form and use a mixture of horizontal and vertical splitters in the same form.
- A splitter can cross (overlap) an object. This object will be resized when the splitter is moved.
- Splitter stops are calculated so that the objects moved remain entirely visible in the form or do not pass under/next to another splitter. When the [Pusher](#) property is associated with a splitter, its movement to the right or downward does not encounter any stops.
- If you resize a form using a splitter, the new dimensions of the form are saved only while the form is being displayed. Once a form is closed, the initial dimensions are restored.

Once it is inserted, the splitter appears as a line. You can modify its [border style](#) to obtain a thinner line or [change its color](#).

### JSON Example:

```
"mySplitter": {  
    "type": "splitter",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20,  
    "splitterMode": "move" //pusher  
}
```

### Supported Properties

[Border Line Style](#) - [Bottom](#) - [Class](#) - [Height](#) - [Help Tip](#) - [Horizontal Sizing](#) - [Left](#) - [Line Color](#) - [Object Name](#) - [Pusher](#) - [Right](#) - [Top](#) - [Type](#) - [Vertical Sizing](#) - [Variable or Expression](#) - [Visibility](#) - [Width](#)

## Interaction with the properties of neighboring objects

In a form, splitters interact with the objects that are around them according to these objects' resizing options:

		<b>Object(s) above an horizontal splitter or to the left of a vertical splitter (1)</b>	<b>Object(s) below an horizontal <i>non-Pusher</i> splitter or to the right of a vertical <i>non-Pusher</i> splitter</b>	<b>Object(s) below an horizontal <i>Pusher</i> splitter or to the right of a vertical <i>Pusher</i> splitter</b>
<b>Resizing options for the object(s)</b>				
None	Remain as is		Are moved with the splitter (position relative to the splitter is not modified) until the next stop. The stop when moving to the bottom or right is either the window's border, or another splitter.	Are moved with the splitter (position relative to the splitter is not modified) indefinitely. No stop is applied (see the next paragraph)
Resize		Keep original position(s), but are resized according to the splitter's new position		
Move		Are moved with the splitter		

(1) You cannot drag the splitter past the right (horizontal) or bottom (vertical) side of an object located in this position.

An object completely contained in the rectangle that defines the splitter is moved at the same time as the splitter.

## Managing splitters programmatically

You can associate an object method with a splitter and it will be called with the `On Clicked` event throughout the entire movement.

A [variable](#) of the *Longint* type is associated with each splitter. This variable can be used in your object and/or form methods. Its value indicates the splitter's current position, in pixels, in relation to its initial position.

- If the value is negative: the splitter was moved toward the top or toward the left,
- If the value is positive: the splitter was moved toward the bottom or toward the right,
- If the value is 0: the splitter was moved to its original position.

You can also move the splitter programmatically: you just have to set the value of the associated variable. For example, if a vertical splitter is associated with a variable named `split1`, and if you execute the following statement: `split1:=-10`, the splitter will be moved 10 pixels to the left — as if the user did it manually. The move is actually performed at the end of the execution of the form or object method containing the statement.

## Static picture

Static pictures are [static objects](#) that can be used for various purposes in 4D forms, including decoration, background, or user interface:



Static pictures are stored outside the forms and inserted by reference. In the form editor, static picture objects are created by copy/paste or drag and drop operations.

If you place a static picture on page 0 of a multi-page form, it will appear automatically as a background element on all pages. You can also include it in an inherited form, applied in the background of other different forms. Either way, your application will run faster than if the picture was pasted into each page.

## Format and location

The original picture must be stored in a format managed natively by 4D (4D recognizes the main picture formats: JPEG, PNG, BMP, SVG, GIF, etc.).

Two main locations can be used for static picture path:

- in the **Resources** folder of the project. Appropriate when you want to share static pictures between several forms in the project. In this case, the Pathname is in the "/RESOURCES/¥<picture path>".
- in an image folder (e.g. named **Images**) within the form folder. Appropriate when the static pictures are used only in the form and/or you want to be able to move or duplicate the whole form within the project or different projects. In this case, the Pathname is "<¥picture path>" and is resolved from the root of the form folder.

## Supported Properties

[Bottom](#) - [CSS Class](#) - [Display](#) - [Height](#) - [Horizontal Sizing](#) - [Left](#) - [Object Name](#) - [Pathname](#) - [Right](#) - [Top](#) - [Type](#) - [Vertical Sizing](#) - [Visibility](#) - [Width](#)

## Stepper

A stepper lets the user scroll through numeric values, durations (times) or dates by predefined steps by clicking on the arrow buttons.

Stepper associated with vStep variable



## Using steppers

You can assign the variable associated with the object to an enterable area (field or variable) to store or modify the current value of the object.

A stepper can be associated directly with a number, time or date variable.

- For values of the time type, the Minimum, Maximum and Step properties represent seconds. For example, to set a stepper from 8:00 to 18:00 with 10-minute steps:
  - `minimum` = 28 800 ( $8*60*60$ )
  - `maximum` = 64 800 ( $18*60*60$ )
  - `step` = 600 ( $10*60$ )
- For values of the date type, the value entered in the `step` property represents days. The Minimum and Maximum properties are ignored.

For the stepper to work with a time or date variable, it is imperative to set its type in the form AND to declare it explicitly via the `C_TIME` or `C_DATE` command.

For more information, please refer to [Using indicators](#) in the "Progress Indicator" page.

## Supported Properties

[Border Line Style](#) - [Bottom](#) - [Class](#) - [Enterable](#) - [Execute object method](#) - [Expression Type](#) (only "integer", "number", "date", or "time") - [Height](#) - [Help Tip](#) - [Horizontal Sizing](#) - [Left](#) - [Maximum](#) - [Minimum](#) - [Object Name](#) - [Right](#) - [Step](#) - [Top](#) - [Type](#) - [Variable or Expression](#) - [Vertical Sizing](#) - [Visibility](#) - [Width](#)

## See also

- [progress indicators](#)
- [rulers](#)

## Subform

A subform is a form included in another form.

## Terminology

In order to clearly define the concepts implemented with subforms, here are some definitions for certain terms used:

- **Subform:** a form intended for inclusion in another form, itself called the parent form.
- **Parent form:** a form containing one or more subform(s).
- **Subform container:** an object included in the parent form, displaying an instance of the subform.
- **Subform instance:** the representation of a subform in a parent form. This concept is important because it is possible to display several instances of the same subform in a parent form.
- **List form:** instance of subform displayed as a list.
- **Detail form:** page-type input form associated with a list-type subform that can be accessed by double-clicking in the list.

## List subforms

A list subform lets you enter, view, and modify data in other tables. You usually use list subforms in databases in which you have established One to Many relations. A list subform on a form in a related One table lets you view, enter, and modify data in a related Many table. You can have several subforms coming from different tables in the same form. However, it is not possible to place two subforms that belong to the same table on the same page of a form.

For example, a Contacts manager database might use a list subform to display all the telephone numbers for a particular contact. Although the telephone numbers appear on the Contacts screen, the information is actually stored in a related table. Using a One to Many relation, this database design makes it easy to store an unlimited number of telephone numbers per contact. With automatic relations, you can support data entry directly into the related Many table without programming.

Although list subforms are generally associated with Many tables, a subform instance can display the records of any other database table.

You can also allow the user to enter data in the List form. Depending on the configuration of the subform, the user may display the detail form by double-clicking on a subrecord or by using the commands for adding and editing subrecords.

4D offers three standard actions to meet the basic needs for managing subrecords: `Edit Subrecord`, `Delete Subrecord`, and `Add Subrecord`. When the form includes several subform instances, the action will apply to the subform that has the focus.

## Page subforms

Page subforms can display the data of the current subrecord or any type of pertinent value depending on the context (variables, pictures, and so on). One of the main advantages of using page subforms is that they can include advanced functionalities and can interact directly with the parent form (widgets). Page subforms also have

their own specific properties and events; you can manage them entirely by programming.

The page subform uses the input form indicated by the [Detail Form](#) property. Unlike a list subform, the form used can come from the same table as the parent form. It is also possible to use a project form. When executed, a page subform has the same standard display characteristics as an input form.

4D Widgets are predefined compound objects based upon page subforms. They are described in detail in a separate manual, [4D Widgets](#).

## Using the bound variable or expression

You can bind [a variable or an expression](#) to a subform container object. This is very useful to synchronize values from the parent form and its subform(s).

By default, 4D creates a variable or expression of [object type](#) for a subform container, which allows you to share values in the context of the subform using the [Form](#) command ([see below](#)). However, you can use a variable or expression of any scalar type (time, integer, etc.) especially if you only need to share a single value:

- Define a bound variable or expression of a scalar type and call the `OBJECT Get subform container value` and `OBJECT SET SUBFORM CONTAINER VALUE` commands to exchange values when [On Bound Variable Change](#) or [On Data Change](#) form events occur. This solution is recommended to synchronize a single value.
- Define a bound variable or expression of the **object** type and use the [Form](#) command to access its properties from the subform. This solution is recommended to synchronize several values.

## Synchronizing parent form and subform (single value)

Binding the same variable or expression to your subform container and other objects of the parent form lets you link the parent form and subform contexts to put the finishing touches on sophisticated interfaces. Imagine a subform that contains a clock displaying a static time, inserted into a parent form containing an [input area](#):

In the parent form, both objects (input area and subform container) **have the same value as Variable or Expression**. It can be a variable (e.g. `parisTime`), or an expression (e.g. `Form.parisTime`).

### ⓘ INFO

To display a static time, you must use the appropriate [data type](#) for the [variable or expression](#):

- If you use a variable (e.g. `parisTime`), it must be of the `text` or `time` type.
- If you use an expression (e.g. `Form.myValue`), it must contain a `text` value.

The text value must be formatted "hh:mm:ss".

In the subform, the clock object is managed through the `Form.clockValue` property.

## Updating the contents of a subform

Case 1: The value of the parent form variable or expression is modified and this

modification must be passed on to a subform.

`parisTime` or `Form.parisTime` changes to "12:15:00" in the parent form, either because the user entered it, or because it was updated dynamically (via the `String(Current time)` statement for example). This triggers the [On Bound Variable Change](#) event in the subform's Form method.

The following code is executed:

```
// Subform form method
If (Form event code=On Bound Variable Change) //bound variable or
expression was modified in the parent form
    Form.clockValue:=OBJECT Get subform container value //synchronize
the local value
End if
```

It updates the value of `Form.clockValue` in the subform:

The [On Bound Variable Change](#) form event is generated:

- as soon as a value is assigned to the variable/expression of the parent form, even if the same value is reassigned
- if the subform belongs to the current form page or to page 0.

Note that, as in the above example, it is preferable to use the `OBJECT Get subform container value` command which returns the value of the expression in the subform container rather than the expression itself because it is possible to insert several subforms in the same parent form (for example, a window displaying different time zones contains several clocks).

Modifying the bound variable or expression triggers form events which let you synchronize the parent form and subform values:

- Use the [On Bound Variable Change](#) form event to indicate to the subform (form method of subform) that the variable or expression was modified in the parent form.
- Use the [On Data Change](#) form event to indicate to the subform container that the variable or expression value was modified in the subform.

## Updating the contents of a parent form

Case 2: The contents of the subform are modified and this modification must be passed on to the parent form.

Inside the subform, the button changes the value of the `Form.clockValue` expression of type Text attached to the clock object. This triggers the [On Data Change](#) form event inside the clock object (this event must be selected for the object), which updates the `Form.parisTime` value in the main form.

The following code is executed:

```
// subform clock object method
If (Form event code=On Data Change) //whatever the way the value is
changed
    OBJECT SET SUBFORM CONTAINER VALUE (Form.clockValue) //Push the
value to the container
End if
```

Everytime the value of `Form.clockValue` changes in the subform, `parisTime` or `Form.parisTime` in the subform container is also updated.

If the variable or expression value is set at several locations, 4D uses the value that was loaded last. It applies the following loading order: 1-Object methods of subform, 2-Form method of subform, 3-Object methods of parent form, 4-Form method of parent form

## Synchronizing parent form and subform (multiple values)

By default, 4D binds a variable or expression of [object type](#) to each subform. The contents of this object can be read and/or modified from within the parent form and from the subform, allowing you to share multiple values in a local context.

When bound to the subform container, this object is returned by the `Form` command directly in the subform. Since objects are always passed by reference, if the user modifies a property value in the subform, it will automatically be saved in the object itself and thus, available to the parent form. On the other hand, if a property of the object is modified by the user in the parent form or by programming, it will be automatically updated in the subform. No event management is necessary.

For example, in a subform, inputs are bound to the `Form` object properties (of the subform form):

Lastname:	<code>Form.lastname</code>
Firstname:	<code>Form.firstname</code>

In the parent form, you display the subform twice. Each subform container is bound to an expression which is a property of the `Form` object (of the parent form):

The button only creates `mother` and `father` properties in the parent's `Form` object:

```
//Add values button object method  
Form.mother:=New object("lastname"; "Hotel"; "firstname"; "Anne")  
Form.father:=New object("lastname"; "Golf"; "firstname"; "Félix")
```

When you execute the form and click on the button, you see that all values are correctly displayed:

Father	Golf
Lastname:	Golf
Firstname:	Félix
Mother	Hotel
Lastname:	Hotel
Firstname:	Anne
<b>Add values</b>	

If you modify a value either in the parent form or in the subform, it is automatically updated in the other form because the same object is used:

The image shows two separate windows representing forms in a 4D database.

**Left Window (Father Form):**

- Label: Father
- Text input field: Lastname: Wolf
- Text input field: Firstname: Félix

**Right Window (Mother Form):**

- Label: Mother
- Text input field: Lastname: Hotelle
- Text input field: Firstname: Anne

Two red arrows point from the "Wolf" entry in the Father's last name field to the "Hotelle" entry in the Mother's last name field, indicating that changes made in one form are reflected in the other.

## Using pointers (compatibility)

In versions prior to 4D v19 R5, synchronization between parent forms and subforms was handled through **pointers**. For example, to update a subform object, you could call the following code:

```
// Subform form method
If (Form event code=On Bound Variable Change)
    ptr:=OBJECT Get pointer(Object subform container)
    clockValue:=ptr->
End if
```

**This principle is still supported for compatibility but is now deprecated since it does not allow binding expressions to subforms.** It should no longer be used in your developments. In any cases, we recommend to use the [Form command](#) or the [OBJECT Get subform container value](#) and [OBJECT SET SUBFORM CONTAINER VALUE commands](#) to synchronize form and subform values.

## Advanced inter-form programming

Communication between the parent form and the instances of the subform may require going beyond the exchange of values through the bound variable. In fact, you may want to update variables in subforms according to the actions carried out in the parent form and vice versa. If we use the previous example of the "dynamic clock" type subform, we may want to set one or more alarm times for each clock.

4D has implemented the following mechanisms to meet these needs:

- Calling of a container object from the subform using the [CALL SUBFORM CONTAINER](#) command
- Execution of a method in the context of the subform via the [EXECUTE METHOD IN SUBFORM](#) command

The [GOTO OBJECT](#) command looks for the destination object in the parent form even if it is executed from a subform.

### CALL SUBFORM CONTAINER command

The [CALL SUBFORM CONTAINER](#) command lets a subform instance send an [event](#) to the subform container object, which can then process it in the context of the parent form. The event is received in the container object method. It may be at the origin of any event detected by the subform (click, drag-and-drop, etc.).

The code of the event is unrestricted (for example, 20000 or -100). You can use a code that corresponds to an existing event (for example, 3 for [On Validate](#)), or use a custom code. In the first case, you can only use events that you have checked in the Property List for subform containers. In the second case, the code must not correspond to any existing form event. It is recommended to use a negative value to be sure that this code will not be used by 4D in future versions.

For more information, refer to the description of the [CALL SUBFORM CONTAINER](#) command.

## **EXECUTE METHOD IN SUBFORM command**

The [EXECUTE METHOD IN SUBFORM](#) command lets a form or one of its objects request the execution of a method in the context of the subform instance, which gives it access to the subform variables, objects, etc. This method can also receive parameters.

This mechanism is illustrated in the following diagram:

For more information, refer to the description of the [EXECUTE METHOD IN SUBFORM](#) command.

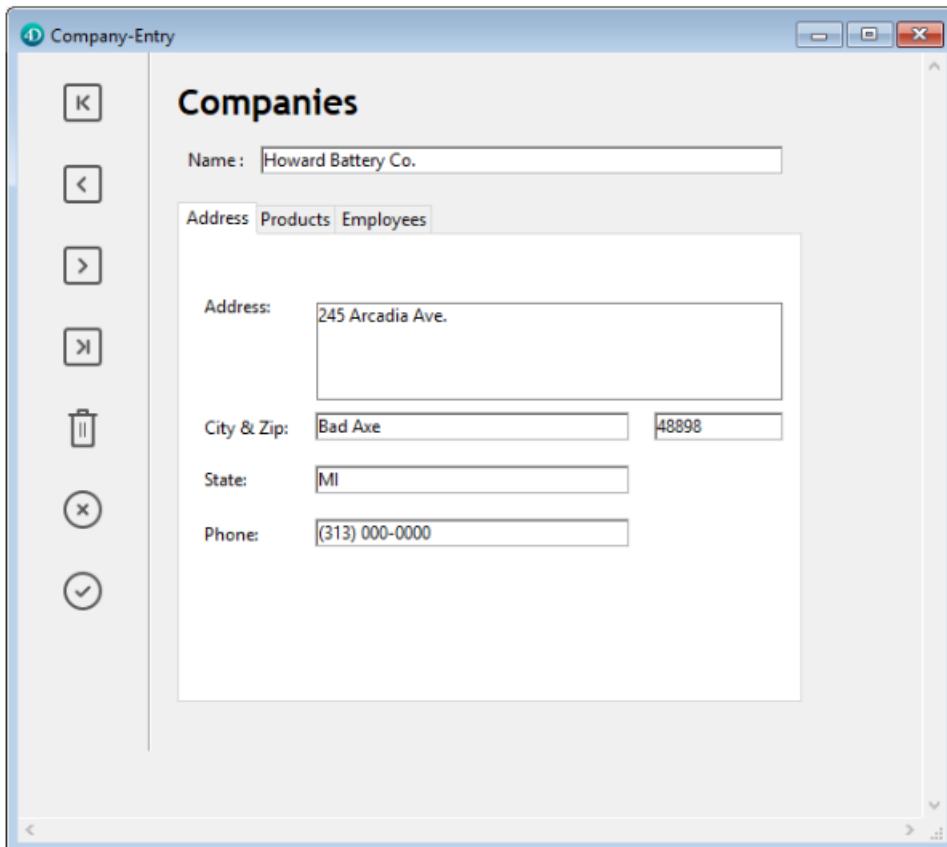
## **Supported Properties**

[Border Line Style](#) - [Bottom](#) - [Class](#) - [Detail Form](#) - [Double click on empty row](#) - [Double click on row](#) - [Enterable in list](#) - [Expression Type](#) - [Focusable](#) - [Height](#) - [Hide focus rectangle](#) - [Horizontal Scroll Bar](#) - [Horizontal Sizing](#) - [Left](#) - [List Form](#) - [Method](#) - [Object Name](#) - [Print Frame](#) - [Right](#) - [Selection mode](#) - [Source](#) - [Top](#) - [Type](#) - [Variable or Expression](#) - [Vertical Scroll Bar](#) - [Vertical Sizing](#) - [Visibility](#) - [Width](#)

## Tab Controls

A tab control creates an object that lets the user choose among a set of virtual screens that are enclosed by the tab control object. Each screen is accessed by clicking its tab.

The following multi-page form uses a tab control object:

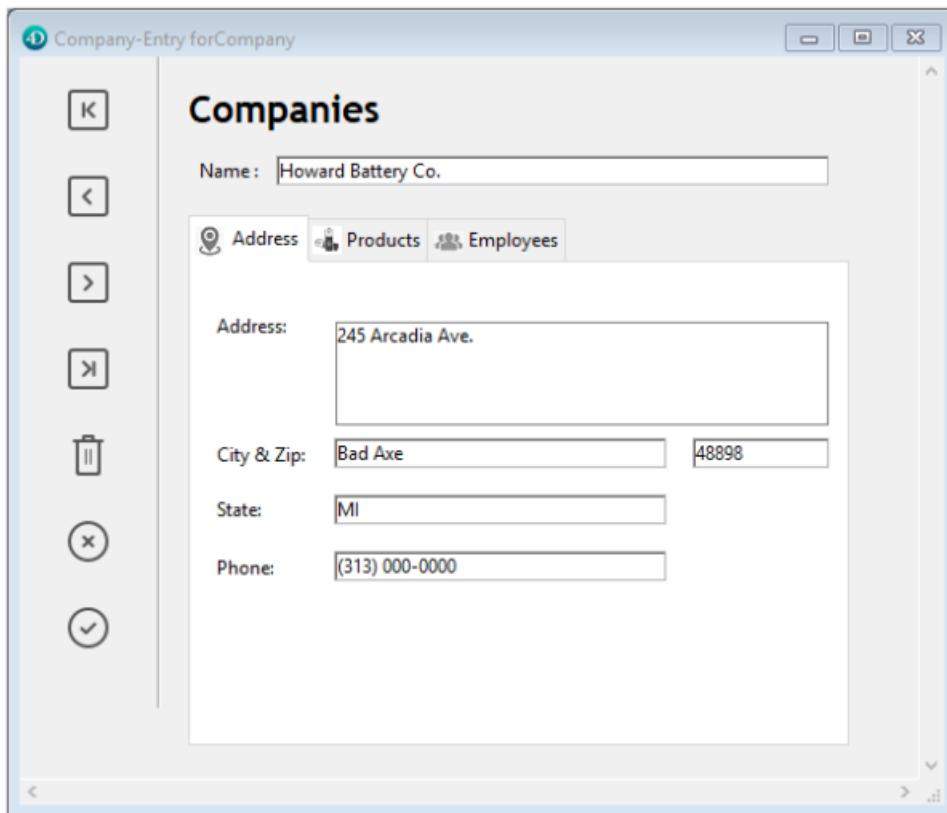


To navigate from screen to screen, the user simply clicks the desired tab.

The screens can represent pages in a multi-page form or an object that changes when the user clicks a tab. If the tab control is used as a page navigation tool, then the [FORM\\_GOTO\\_PAGE](#) command or the `gotoPage` standard action would be used when a user clicks a tab.

Another use of the tab control is to control the data that is displayed in a subform. For example, a Rolodex could be implemented using a tab control. The tabs would display the letters of the alphabet and the tab control's action would be to load the data corresponding to the letter that the user clicked.

Each tab can display labels or labels and a small icon. If you include icons, they appear to the left of each label. Here is an example of a tab control that uses icons:



When you create a tab control, 4D manages the spacing and placement of the tabs. You only need to supply the labels in the form of an array, or the icons and labels in the form of a hierarchical list.

If the tab control is wide enough to display all the tabs with both the labels and icons, it displays both. If the tab control is not wide enough to display both the labels and icons, 4D displays the icons only. If it can't fit all the icons, it places scroll arrows to the right of the last visible tab. The scroll arrows allow the user to scroll the icons to the left or right.

Under macOS, in addition to the standard position (top), the tab controls can also be aligned to the bottom.

### JSON Example:

```
"myTab": {  
    "type": "tab",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20,  
    "labelsPlacement": "bottom" //define the direction  
}
```

## Adding labels to a tab control

To supply the labels for a tab control, you can use:

- an object
- a choice list
- an array

### Using an object

You can assign an [object](#) encapsulating a [collection](#) as the [data source](#) of the tab control. The object must contain the following properties:

Property	Type	Description
values	Collection	Mandatory - Collection of scalar values. Only string values are supported. If invalid, empty or not defined, the tab control is empty
index	number	Index of the currently tab control page (value between 0 and collection.length-1)
currentValue	Text	Currently selected value

The initialization code must be executed before the form is presented to the user.

In the following example, `Form.tabControl` has been defined as tab control [expression](#). You can associate the [gotoPage standard action](#) to the form object:

```
Form.tabControl:=New object
Form.tabControl.values:=New collection("Page 1"; "Page 2"; "Page 3")
Form.tabControl.index:=2 //start on page 3
```

## Using a choice list

You can assign a [choice list](#) to the tab control, either through a collection (static list) or a JSON pointer to a json list ("\$ref"). Icons associated with list items in the Lists editor will be displayed in the tab control.

## Using a Text array

You can create a Text array that contains the names of each page of the form. This code must be executed before the form is presented to the user. For example, you could place the code in the object method of the tab control and execute it when the [On Load](#) event occurs.

```
ARRAY TEXT(arrPages;3)
arrPages{1}:="Name"
arrPages{2}:="Address"
arrPages{3}:="Notes"
```

You can also store the names of the pages in a hierarchical list and use the [LIST TO ARRAY](#) command to load the values into the array.

## Goto page features

### FORM GOTO PAGE command

You can use the [FORM GOTO PAGE](#) command in the tab control's method:

```
FORM GOTO PAGE(arrPages)
```

The command is executed when the [On Clicked](#) event occurs. You should then clear the array when the [On Unload](#) event occurs.

Here is an example object method:

```
Case of
  :(Form event=On Load)
    LIST TO ARRAY("Tab Labels";arrPages)
  :(Form event=On Clicked)
    FORM GOTO PAGE(arrPages)
  :(Form event=On Unload)
    CLEAR VARIABLE(arrPages)
End case
```

## Goto Page action

When you assign the `gotoPage` [standard action](#) to a tab control, 4D will automatically display the page of the form that corresponds to the number of the tab that is selected.

For example, if the user selects the 3rd tab, 4D will display the third page of the current form (if it exists).

## Supported Properties

[Bold](#) - [Bottom](#) - [Choice List](#) - [Class](#) - [Expression Type](#) - [Font](#) - [Font Size](#) - [Height](#) - [Help Tip](#) - [Horizontal Sizing](#) - [Italic](#) - [Left](#) - [Object Name](#) - [Right](#) - [Save value](#) - [Standard action](#) - [Tab Control Direction](#) - [Top](#) - [Type](#) - [Underline](#) - [Vertical Sizing](#) - [Variable or Expression](#) - [Visibility](#) - [Width](#)

## Text

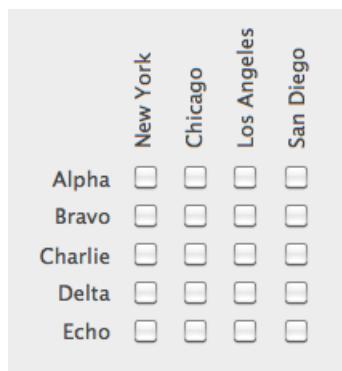
A text object allows you to display static written content (e.g., instructions, titles, labels, etc.) on a form. These static text areas can become dynamic when they include dynamic references. For more information, refer to [Using references in static text](#).

### JSON Example:

```
"myText": {  
    "type": "text",  
    "text": "Hello World!",  
    "textAlign": "center",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20,  
    "stroke": "#ff0000"      //text color  
    "fontWeight": "bold"  
}
```

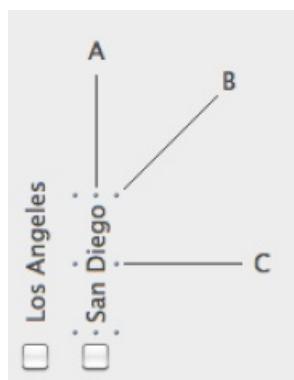
## Rotation

4D lets you rotate text areas in your forms using the [Orientation](#) property.



Text rotation can be defined for a process using the `OBJECT SET TEXT ORIENTATION` language command.

Once a text is rotated, you can still change its size or position, as well as all its properties. Note that the text area's height and width properties do not depend on its orientation:



- If the object is resized in direction A, its `width` is modified;
- If the object is resized in direction C, its `height` is modified;
- If the object is resized in direction B, both its `width` and `height` are modified.

## Supported Properties

- History

[Bold](#) - [Border Line Style](#) - [Bottom](#) - [Class](#) - [Corner radius](#) - [Fill Color](#) - [Font](#) - [Font Color](#) - [Font Size](#) - [Height](#) - [Horizontal Alignment](#) - [Horizontal Sizing](#) - [Italic](#) - [Left](#) - [Object Name](#) - [Orientation](#) - [Right](#) - [Title](#) - [Top](#) - [Type](#) - [Underline](#) - [Vertical Sizing](#) - [Visibility](#) - [Width](#)

## Web Area

Web areas can display various types of web content within your forms: HTML pages with static or dynamic contents, files, pictures, JavaScript, etc. The rendering engine of the web area depends on the execution platform of the application and the selected [rendering engine option](#).

It is possible to create several web areas in the same form. Note, however, that the use of web areas must follow [several rules](#).

Several dedicated [standard actions](#), numerous [language commands](#) as well as generic and specific [form events](#) allow the developer to control the functioning of web areas. Specific variables can be used to exchange information between the area and the 4D environment.

## Specific properties

### Associated variables

Two specific variables can be associated with each web area:

- [URL](#) --to control the URL displayed by the web area
- [Progression](#) -- to control the loading percentage of the page displayed in the web area.

As of 4D v19 R5, the Progression variable is no longer updated in Web Areas using the [Windows system rendering engine](#).

### Web rendering engine

You can choose between [two rendering engines](#) for the web area, depending on the specifics of your application.

Selecting the embedded web rendering engine allows you to call 4D methods from the web area and to make sure features on macOS and Windows are similar. Selecting the system rendering engine is recommended when the web area is connected to the Internet because it always benefits from the latest security updates.

### Access 4D methods

When the [Access 4D methods](#) property is selected, you can call 4D methods from a web area.

This property is only available if the web area [uses the embedded web rendering engine](#).

### \$4d object

The [4D embedded web rendering engine](#) supplies the area with a JavaScript object named \$4d that you can associate with any 4D project method using the "." object notation.

For example, to call the [HelloWorld](#) 4D method, you just execute the following statement:

```
$4d.HelloWorld();
```

JavaScript is case sensitive so it is important to note that the object is named \$4d (with a lowercase "d").

The syntax of calls to 4D methods is as follows:

```
$4d.4DMethodName(param1,paramN,function(result){})
```

- `param1...paramN`: You can pass as many parameters as you need to the 4D method. These parameters can be of any type supported by JavaScript (string, number, array, object).
- `function(result)`: Function to pass as last argument. This "callback" function is called synchronously once the 4D method finishes executing. It receives the `result` parameter.
- `result`: Execution result of the 4D method, returned in the "\$0" expression. This result can be of any type supported by JavaScript (string, number, array, object). You can use the `C_OBJECT` command to return the objects.

By default, 4D works in UTF-8. When you return text containing extended characters, for example characters with accents, make sure the encoding of the page displayed in the Web area is declared as UTF-8, otherwise the characters may be rendered incorrectly. In this case, add the following line in the HTML page to declare the encoding: `<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />`

## Example 1

Given a 4D project method named `today` that does not receive parameters and returns the current date as a string.

4D code of `today` method:

```
C_TEXT($0)
$0:=String(Current date;System date long)
```

In the web area, the 4D method can be called with the following syntax:

```
$4d.today()
```

The 4D method does not receive any parameters but it does return the value of \$0 to the callback function called by 4D after the execution of the method. We want to display the date in the HTML page that is loaded by the web area.

Here is the code of the HTML page:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<script type="text/javascript">
$4d.today(function(dollarZero)
{
    var curDate = dollarZero;
    document.getElementById("mydiv").innerHTML=curDate;
});
</script>
</head>
<body>Today is: <div id="mydiv"></div>
</body>
</html>
```

## Example 2

The 4D project method `calcSum` receives parameters (`$1...$n`) and returns their sum in `$0`:

4D code of `calcSum` method:

```
C_REAL(${1}) // receives n REAL type parameters
C_REAL($0) // returns a Real
C_LONGINT($i;$n)
$n:=Count parameters
For($i;1;$n)
    $0:=$0+$i
End for
```

The JavaScript code run in the web area is:

```
$4d.calcSum(33, 45, 75, 102.5, 7, function(dollarZero)
{
    var result = dollarZero // result is 262.5
});
```

## Standard actions

Four specific standard actions are available for managing web areas automatically:

`Open Back URL`, `Open Forward URL`, `Refresh Current URL` and `Stop Loading URL`. These actions can be associated with buttons or menu commands and allow quick implementation of basic web interfaces. These actions are described in [Standard actions](#).

## Form events

Specific form events are intended for programmed management of web areas, more particularly concerning the activation of links:

- [On Begin URL Loading](#)
- [On URL Resource Loading](#)
- [On End URL Loading](#)
- [On URL Loading Error](#)
- [On URL Filtering](#)
- [On Open External Link](#)
- [On Window Opening Denied](#)

In addition, web areas support the following generic form events:

- [On Load](#)
- [On Unload](#)
- [On Getting Focus](#)
- [On Losing Focus](#)

## Web area rules

### User interface

When the form is executed, standard browser interface functions are available to the user in the web area, which permit interaction with other form areas:

- **Edit menu commands:** When the web area has the focus, the **Edit** menu

commands can be used to carry out actions such as copy, paste, select all, etc., according to the selection.

- **Context menu:** It is possible to use the standard [context menu](#) of the system with the web area. Display of the context menu can be controlled using the `WA SET PREFERENCE` command.
- **Drag and drop:** The user can drag and drop text, pictures and documents within the web area or between a web area and the 4D form objects, according to the 4D object properties. For security reasons, changing the contents of a web area by means of dragging and dropping a file or URL is not allowed by default. In this case, the cursor displays a "forbidden" icon . You have to use the `WA SET PREFERENCE (*;"warea";WA enable URL drop;True)` statement to display a "drop" icon and generate the [On Window Opening Denied](#) event. In this event, you can call the [WA OPEN URL](#) command or set the [URL variable](#) in response to a user drop.

Drag and drop features described above are not supported in web areas using the [macOS system rendering engine](#).

## Subforms

For reasons related to window redrawing mechanisms, the insertion of a web area into a subform is subject to the following constraints:

- The subform must not be able to scroll
- The limits of the web area must not exceed the size of the subform

Superimposing a web area on top of or beneath other form objects is not supported.

## Web Area and Web server conflict (Windows)

In Windows, it is not recommended to access, via a web area, the Web server of the 4D application containing the area because this configuration could lead to a conflict that freezes the application. Of course, a remote 4D can access the Web server of 4D Server, but not its own web server.

## Insertion of protocol (macOS)

The URLs handled by programming in web areas in macOS must begin with the protocol. For example, you need to pass the string "<http://www.mysite.com>" and not just "[www.mysite.com](http://www.mysite.com)".

## Web inspector

You can view and use a web inspector within web areas in your forms or in offscreen web areas. The web inspector is a debugger which allows parsing the code and the flow of information of the web pages.

To display the Web inspector, you can either execute the `WA OPEN WEB INSPECTOR` command, or use the context menu of the web area.

- **Execute the `WA OPEN WEB INSPECTOR` command**  
This command can be used directly with onscreen (form object) and offscreen web areas.
- **Use the web area context menu**  
This feature can only be used with onscreen web areas and requires that the following conditions are met:

- the [context menu](#) for the web area is enabled
- the use of the inspector is expressly enabled in the area by means of the following statement:  
`WA SET PREFERENCE (*;"WA";WA enable Web inspector;True)`

With [Windows system rendering engine](#), a change in this preference requires a navigation action in the area (for example, a page refresh) to be taken into account.

For more information, refer to the description of the `WA SET PREFERENCE` command.

When you have done the settings as described above, you then have new options such as **Inspect Element** in the context menu of the area. When you select this option, the web inspector window is displayed.

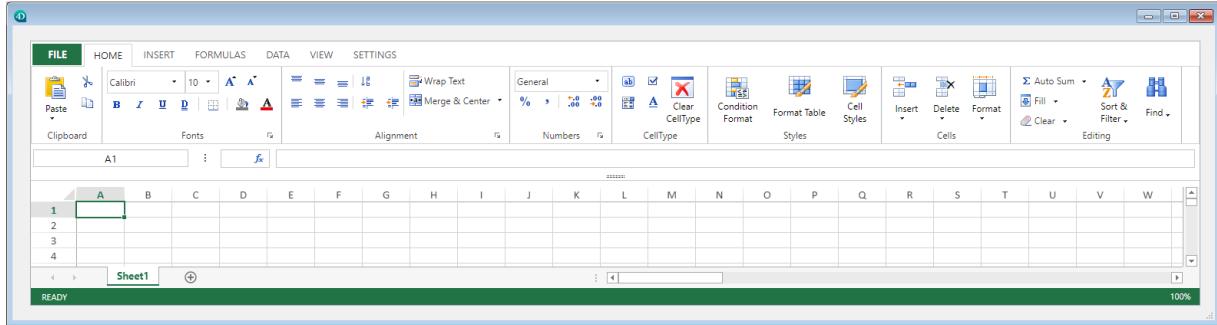
For a detailed description of the features of this debugger, refer to the documentation provided by the web rendering engine.

## Supported Properties

[Border Line Style](#) - [Bottom](#) - [Class](#) - [Context Menu](#) - [Height](#) - [Horizontal Sizing](#) - [Left](#) - [Method](#) - [Object Name](#) - [Progression](#) - [Right](#) - [Top](#) - [Type](#) - [URL](#) - [Use embedded Web rendering engine](#) - [Variable or Expression](#) - [Vertical Sizing](#) - [Visibility](#) - [Width](#)

## 4D View Pro area

4D View Pro allows you to insert and display a spreadsheet area in your 4D forms. A spreadsheet is an application containing a grid of cells into which you can enter information, execute calculations, or display pictures.



Once you use 4D View Pro areas in your forms, you can import and export spreadsheets documents.

## Using 4D View Pro areas

4D View Pro areas are documented in the [4D View Pro section](#).

## Supported Properties

[Border Line Style](#) - [Bottom](#) - [Class](#) - [Height](#) - [Horizontal Sizing](#) - [Left](#) - [Method](#) - [Object Name](#) - [Right](#) - [Show Formula Bar](#) - [Type](#) - [User Interface](#) - [Vertical Sizing](#) - [Visibility](#) - [Width](#)

## 4D Write Pro area

4D Write Pro offers 4D users an advanced word-processing tool, fully integrated with your 4D application. Using 4D Write Pro, you can write pre-formatted emails and/or letters containing images, a scanned signature, formatted text and placeholders for dynamic variables. You can also create invoices or reports dynamically, including formatted text and images.

## Using 4D Write Pro areas

4D Write Pro areas are documented in the [4D Write Pro Reference](#) manual.

## Supported Properties

[Auto Spellcheck](#) - [Border Line Style](#) - [Bottom](#) - [Class](#) - [Context Menu](#) - [Draggable](#) - [Droppable](#) - [Enterable](#) - [Focusable](#) - [Height](#) - [Hide focus rectangle](#) - [Horizontal Scroll Bar](#) - [Horizontal Sizing](#) - [Keyboard Layout](#) - [Left](#) - [Method](#) - [Object Name](#) - [Print](#) - [Variable Frame](#) - [Resolution](#) - [Right](#) - [Selection always visible](#) - [Show background](#) - [Show footers](#) - [Show headers](#) - [Show hidden characters](#) - [Show horizontal ruler](#) - [Show HTML WYSIWYG](#) - [Show page frame](#) - [Show references](#) - [Show vertical ruler](#) - [Type](#) - [Vertical Sizing](#) - [Vertical Scroll Bar](#) - [View mode](#) - [Visibility](#) - [Width](#) - [Zoom](#)

# Form object JSON property list

You will find in this page a comprehensive list of all object properties sorted through their JSON name. Click on a property name to access its detailed description.

In the "Form Object Properties" chapter, properties are sorted according the Property List names and themes.

[a](#) - [b](#) - [c](#) - [d](#) - [e](#) - [f](#) - [g](#) - [h](#) - [i](#) - [j](#) - [k](#) - [l](#) - [m](#) - [n](#) - [p](#) - [r](#) - [s](#) - [t](#) - [u](#) - [v](#) - [w](#) - [z](#)

Property	Description	Possible Values
<a href="#">a</a>		
<a href="#">action</a>	Typical activity to be performed.	The name of a valid standard action.
<a href="#">allowFontColorPicker</a>	Allows displaying system font picker or color picker to edit object attributes	true, false (default)
<a href="#">alternateFill</a>	Allows setting a different background color for odd-numbered rows/columns in a list box.	Any CSS value; "transparent"; "automatic"; "automaticAlternate"
<a href="#">automaticInsertion</a>	Enables automatically adding a value to a list when a user enters a value that is not in the object's associated choice list.	true, false
<a href="#">b</a>		
<a href="#">booleanFormat</a>	Specifies only two possible values.	true, false
<a href="#">borderRadius</a>	The radius value for round rectangles.	minimum: 0
<a href="#">borderStyle</a>	Allows setting a standard style for the object border.	"system", "none", "solid", "dotted", "raised", "sunken", "double"
<a href="#">bottom</a>	Positions an object at the bottom (centered).	minimum: 0
<a href="#">c</a>		
<a href="#">choiceList</a>	A list of choices associated with an object	A list of choices
<a href="#">class</a>	A list of space-separated words used as class selectors in css files.	A list of class names
<a href="#">columnCount</a>	Number of columns.	minimum: 1
<a href="#">columns</a>	A collection of list box columns	Collection of column objects with defined column properties

Property	Description	Possible Values
<a href="#">contextMenu</a>	Provides the user access to a standard context menu in the selected area.	"automatic", "none"
<a href="#">continuousExecution</a>	Designates whether or not to run the method of an object while the user is tracking the control.	true, false
<a href="#">controlType</a>	Specifies how the value should be rendered in a list box cell.	"input", "checkbox" (for boolean / numeric columns), "automatic", "popup" (only for boolean columns)
<a href="#">currentItemSource</a>	The last selected item in a list box.	Object expression
<a href="#">currentItemPositionSource</a>	The position of the last selected item in a list box.	Number expression
<a href="#">customBackgroundPicture</a>	Sets the picture that will be drawn in the background of a button.	Relative path in POSIX syntax. Must be used in conjunction with the style property with the "custom" option.
<a href="#">customBorderX</a>	Sets the size (in pixels) of the internal horizontal margins of an object. Must be used with the style property with the "custom" option.	minimum: 0
<a href="#">customBorderY</a>	Sets the size (in pixels) of the internal vertical margins of an object. Must be used with the style property with the "custom" option.	minimum: 0
<a href="#">customOffset</a>	Sets a custom offset value in pixels. Must be used with the style property with the "custom" option.	minimum: 0
<a href="#">customProperties</a>	Advanced properties (if any)	JSON string or base64 encoded string
<b>d</b>		
<a href="#">dataSource</a> (objects)	Specifies the source of the data.	
<a href="#">dataSource</a> (subforms)		
<a href="#">dataSource</a> (array list box)		
<a href="#">dataSource</a> (Collection or entity selection list box)		A 4D variable, field name, or an arbitrary complex language expression.
<a href="#">dataSource</a> (list box column)		
<a href="#">dataSource</a> (hierarchical list box)		
<a href="#">dataSourceTypeHint</a> (objects)		"integer", "boolean", "number", "picture", "text", "date", "time", "arrayText", "arravDate", "arravTime".
<a href="#">dataSourceTypeHint</a> (list box type)		

<b>Property</b>	<b>Description</b>	<b>Possible Values</b>
<a href="#">dateFormat</a>	Controls the way dates appear when displayed or printed. Must only be selected among the 4D built-in formats.	"array", "possibleCollection", "object", "undefined", "systemShort", "systemMedium", "systemLong", "iso8601", "rfc822", "short", "shortCentury", "abbreviated", "long", "blankIfNull" (can be combined with the other possible values)
<a href="#">defaultButton</a>	Modifies a button's appearance in order to indicate the recommended choice to the user.	true, false
<a href="#">defaultValue</a>	Defines a value or a stamp to be entered by default in an input object	String or "#D", "#H", "#N"
<a href="#">deletableInList</a>	Specifies if the user can delete subrecords in a list subform	true, false
<a href="#">detailForm</a> (list box) <a href="#">detailForm</a> (subform)	Associates a detail form with a list subform.	Name (string) of table or project form, a POSIX path (string) to a .json file describing the form, or an object describing the form
<a href="#">display</a>	The object is drawn or not on the form.	true, false
<a href="#">doubleClickInEmptyAreaAction</a>	Action to perform in case of a double-click "addSubrecord" or "" to do on an empty line of a nothing list subform.	"addSubrecord" or ""
<a href="#">doubleClickInRowAction</a> (list box) <a href="#">doubleClickInRowAction</a> (subform)	Action to perform in case of a double-click on a record.	"editSubrecord", "displaySubrecord"
<a href="#">dpi</a>	Screen resolution for the 4D Write Pro area contents.	0=automatic, 72, 96
<a href="#">dragging</a>	Enables dragging function.	"none", "custom", "automatic" (excluding list, list box)
<a href="#">dropping</a>	Enables dropping function.	"none", "custom", "automatic" (excluding list, list box)
<b>e</b>		
<a href="#">enterable</a>	Indicates whether users can enter values into the object.	true, false
<a href="#">enterableInList</a>	Indicates whether users can modify record data directly in the list subform.	true, false
	Associates an entry	

<b>Property</b>	<b>Description</b>	<b>Possible Values</b>
<a href="#">entryFilter</a>	filter with the object or column cells. This property is not accessible if the Enterable property is not enabled.	Text to narrow entries
<a href="#">events</a>	List of all events selected for the object or form	Collection of event names, e.g. ["onClick","onDataChange"...].
<a href="#">excludedList</a>	Allows setting a list whose values cannot be entered in the column.	A list of values to be excluded.
<b>f</b>		
<a href="#">fill</a>	Defines the background color of an object.	Any CSS value, "transparent", "automatic"
<a href="#">focusable</a>	Indicates whether the object can have the focus (and can thus be activated by the keyboard for instance)	true, false
<a href="#">fontFamily</a>	Specifies the name of font family used in the object.	CSS font family name
<a href="#">fontSize</a>	Sets the font size in points when no font theme is selected	minimum: 0
<a href="#">fontStyle</a>	Sets the selected text to slant slightly to the right.	"normal", "italic"
<a href="#">fontTheme</a>	Sets the automatic style	"normal", "main", "additional"
<a href="#">fontWeight</a>	Sets the selected text to appear darker and heavier.	"normal", "bold"
<a href="#">footerHeight</a>	Used to set the row height	positive decimal + px   em
<a href="#">frameDelay</a>	Enables cycling through the contents of the picture button at the specified speed (in ticks).	minimum: 0
<b>g</b>		
<a href="#">graduationStep</a>	Scale display measurement.	minimum: 0
<b>h</b>		
<a href="#">header</a>	Defines the header of a list box column	Object with properties "text", "name", "icon", "dataSource", "fontWeight", "fontStyle", "tooltip"
<a href="#">headerHeight</a>	Used to set the row height	positive decimal + px   em
<a href="#">height</a>	Designates an object's vertical size	minimum: 0

Property	Description	Possible Values
<a href="#">hideExtraBlankRows</a>	Deactivates the visibility of extra, empty rows.	true, false
<a href="#">hideFocusRing</a>	Hides the selection rectangle when the object has the focus.	true, false
<a href="#">hideSystemHighlight</a>	Used to specify hiding highlighted records in the list box.	true, false
<a href="#">highlightSet</a>	string Defines the color of the horizontal lines in Any CSS value, "transparent", a list box (gray by default).	Name of the set.
<a href="#">horizontalLineStroke</a>		
<b>i</b>		
<a href="#">icon</a>	The pathname of the picture used for buttons, check boxes, radio buttons, list box headers.	Relative or filesystem path in POSIX syntax.
<a href="#">iconFrames</a>	Sets the exact number of states present in the picture.	minimum: 1
<a href="#">iconPlacement</a>	Designates the placement of an icon in relation to the form object.	"none", "left", "right"
<a href="#">imageHugsTitle</a>	Defines whether the title and the picture of the button should be visually adjoined.	true (default), false
<b>k</b>		
<a href="#">keyboardDialect</a>	To associate a specific keyboard layout to an input.	A keyboard code string, e.g. "ar-ma"
<b>l</b>		
<a href="#">labels</a>	A list of values to be used as tab control labels	ex: "a", "b", "c", ...
<a href="#">labelsPlacement</a> (objects)	Specifies the location of an object's displayed text.	"none", "top", "bottom", "left", "right"
<a href="#">labelsPlacement</a> (tab control)	Mode for displaying the 4D Write Pro document in the form area.	"page", "draft", "embedded"
<a href="#">layoutMode</a>	Positions an object on the left.	minimum: 0
<a href="#">left</a>	A list of choices associated with a hierarchical list	A list of choices
<a href="#">list</a> , see <a href="#">choiceList</a>	The list box data source	"array", "currentSelection", "namedSelection", "collection"
<a href="#">listboxType</a>		

Property	Source	Description	Possible Values
<a href="#">listForm</a>	NamedSelection / connection	List form to use in the subform.	Name (string) of table or project form, a POSIX path (string) to a .json file describing the form, or an object describing the form
<a href="#">lockedColumnCount</a>		Number of columns that must stay permanently displayed in the left part of a list box.	minimum: 0
<a href="#">loopBackToFirstFrame</a>		Pictures are displayed in a continuous loop.	true, false
<b>m</b>			
<a href="#">max</a>		The maximum allowed value. For numeric steppers, these properties represent seconds when the object is associated with a time type value and are ignored when it is associated with a date type value.	minimum: 0 (for numeric data types)
<a href="#">maxWidth</a>		Designates the largest size allowed for list box columns.	minimum: 0
<a href="#">metaSource</a>		A meta object containing style and selection settings.	An object expression
<a href="#">method</a>		A project method name.	The name of an existing project method
<a href="#">methodsAccessibility</a>		Which 4D methods can be called from a Web area	"none" (default), "all"
<a href="#">min</a>		The minimum allowed value. For numeric steppers, these properties represent seconds when the object is associated with a time type value and are ignored when it is associated with a date type value.	minimum: 0 (for numeric data types)
<a href="#">minWidth</a>		Designates the smallest size allowed for list box columns.	minimum: 0
<a href="#">movableRows</a>		Authorizes the movement of rows during execution.	true, false
<a href="#">multiline</a>		Handles multiline contents.	"yes", "no", "automatic"
<b>n</b>			

<b>Property</b>	<b>Description</b>	<b>Possible Values</b>
<a href="#">name</a>	The name of the form object. (Optional for the form)	Any name which does not belong to an already existing object
<a href="#">numberFormat</a>	Controls the way the alphanumeric fields and variables appear when displayed or printed.	Numbers (including a decimal point or minus sign if necessary)
<b>p</b>		
<a href="#">picture</a>	The pathname of the picture for picture buttons, picture pop-up menus, or static pictures	Relative or filesystem path in POSIX syntax, or "var:\$<variableName>" for picture variable.
<a href="#">pictureFormat</a> (input, list box column or footer) <a href="#">pictureFormat</a> (static picture)	Controls how pictures appear when displayed or printed.	"truncatedTopLeft", "scaled", "truncatedCenter", "tiled", "proportionalTopLeft" (excluding static pictures), "proportionalCenter" (excluding static pictures)
<a href="#">placeholder</a>	Grays out text when the data source value is empty.	Text to be grayed out.
<a href="#">pluginAreaKind</a>	Describes the type of plug-in.	The type of plug-in.
<a href="#">popupPlacement</a>	Allows displaying a symbol that appears as a triangle in the button, which indicates that there is a pop-up menu attached.	"None", Linked", "Separated"
<a href="#">printFrame</a>	Print mode for objects whose size can vary from one record to another depending on their contents	"fixed", "variable", (subform only) "fixedMultiple"
<a href="#">progressSource</a>	A value between 0 and 100, representing the page load completion percentage in the Web area.	minimum: 0
<b>r</b>		
<a href="#">radioGroup</a>	Enables radio buttons to be used in coordinated sets: only one button at a time can be selected in the set.	Radio group name

Property	Description	Possible Values
<a href="#">requiredList</a>	Allows the list where only certain values can be inserted.	A list of mandatory values.
<a href="#">resizable</a>	Designates if the size of an object can be modified by the user.	"true", "false"
<a href="#">resizingMode</a>	Specifies if a list box column should be automatically resized	"rightToLeft", "legacy"
<a href="#">right</a>	Positions an object on the right.	minimum: 0
<a href="#">rowControlSource</a>	A 4D array defining the list box rows.	Array
<a href="#">rowCount</a>	Sets the number of rows.	minimum: 1
<a href="#">rowFillSource</a> (array list box) <a href="#">rowFillSource</a> (selection or collection list box)	The name of an array or expression to apply a custom background color to each row of a list box.	The name of an array or expression.
<a href="#">rowHeight</a>	Sets the height of list box rows.	CSS value unit "em" or "px" (default)
<a href="#">rowHeightAuto</a>	boolean	"true", "false"
<a href="#">rowHeightAutoMax</a>	Designates the largest height allowed for list box rows.	CSS value unit "em" or "px" (default). minimum: 0
<a href="#">rowHeightAutoMin</a>	Designates the smallest height allowed for list box rows.	CSS value unit "em" or "px" (default). minimum: 0
<a href="#">rowHeightSource</a>	An array defining different heights for the rows in a list box.	Name of a 4D array variable.
<a href="#">rowStrokeSource</a> (array list box) <a href="#">rowStrokeSource</a> (selection or collection/entity selection list box)	An array or expression for managing row colors.	Name of array or expression.
<a href="#">rowStyleSource</a> (array list box) <a href="#">rowStyleSource</a> (selection or collection/entity selection list box)	An array or expression for managing row styles.	Name of array or expression.
<b>s</b>		
<a href="#">saveAs</a> (list box column) <a href="#">saveAs</a> (drop-down list)	The type of contents to save in the field or variable associated to the form object	"value", "reference"
<a href="#">scrollbarHorizontal</a>	A tool allowing the user to move the viewing area to the left or right.	"visible", "hidden", "automatic"
<a href="#">scrollbarVertical</a>	A tool allowing the user to move the viewing area up or down.	"visible", "hidden", "automatic"

<b>Property</b>	<b>Description</b>	<b>Possible Values</b>
<a href="#">selectedItemsSource</a>	Collection of selected items in a list box.	Collection expression
<a href="#">selectionMode</a> (hierarchical list)	Allows the selection of multiple records/rows.	"multiple", "single", "none"
<a href="#">selectionMode</a> (list box)		
<a href="#">selectionMode</a> (subform)		
<a href="#">shortcutAccel</a>	Specifies the system to use, Windows or Mac.	true, false
<a href="#">shortcutAlt</a>	Designates the Alt key	true, false
<a href="#">shortcutCommand</a>	Designates the Command key (macOS)	true, false
<a href="#">shortcutControl</a>	Designates the Control key (Windows)	true, false
<a href="#">shortcutKey</a>	The letter or name of a special meaning key.	"[F1]" -> "[F15]", "[Return]", "[Enter]", "[Backspace]", "[Tab]", "[Esc]", "[Del]", "[Home]", "[End]", "[Help]", "[Page up]", "[Page down]", "[left arrow]", "[right arrow]", "[up arrow]", "[down arrow]"
<a href="#">shortcutShift</a>	Designates the Shift key	true, false
<a href="#">showFooters</a>	Displays or hides column footers.	true, false
<a href="#">showGraduations</a>	Displays/Hides the graduations next to the labels.	true, false
<a href="#">showHeaders</a>	Displays or hides column headers.	true, false
<a href="#">showHiddenChars</a>	Displays/hides invisible characters.	true, false
<a href="#">showHorizontalRuler</a>	Displays/hides the horizontal ruler when the document view is in Page view mode	true, false
<a href="#">showHTMLWysiwyg</a>	Enables/disables the HTML WYSIWYG view	true, false
<a href="#">showPageFrames</a>	Displays/hides the page frame when the document view is in Page view mode	true, false
<a href="#">showReferences</a>	Displays all 4D expressions inserted in the 4D Write Pro document as references	true, false
<a href="#">showSelection</a>	Keeps the selection visible within the object after it has lost the focus	true, false

Property	Description	Possible Values
<a href="#">showVerticalRuler</a>	Displays vertical ruler when the document view is in Page view mode	true, false
<a href="#">singleClickEdit</a>	Enables direct passage to edit mode.	true, false
<a href="#">sizingX</a>	Specifies if the horizontal size of an object should be moved or resized when a user resizes the form.	"grow", "move", "fixed"
<a href="#">sizingY</a>	Specifies if the vertical size of an object should be moved or resized when a user resizes the form.	"grow", "move", "fixed"
<a href="#">sortable</a>	Allows sorting column data by clicking the header.	true, false
<a href="#">spellcheck</a>	Activates the spell-check for the object	true, false
<a href="#">splitterMode</a>	When a splitter object has this property, other objects to its right (vertical splitter) or below it (horizontal splitter) are pushed at the same time as the splitter, with no stop.	"grow", "move", "fixed"
<a href="#">startPoint</a>	Starting point for drawing a line object (only available in JSON Grammar).	"bottomLeft", topLeft"
<a href="#">staticColumnCount</a>	Number of columns that cannot be moved during execution.	minimum: 0
<a href="#">step</a>	Minimum interval accepted between values during use. For numeric steppers, this property represents seconds when the object is associated with a time type value and days when it is associated with a date type value.	minimum: 1
<a href="#">storeStyleTags</a>	Store the style tags with the text, even if true false	

<u>Property</u>	<u>Description</u>	<u>Possible Values</u>
<u>stroke</u> (text)	no modification has been made	true, false
<u>stroke</u> (lines)	Specifies the color of the font or line used in the object.	Any CSS value, "transparent", "automatic"
<u>stroke</u> (list box)		
<u>strokeDashArray</u>	Describes dotted line type as a sequence of black and white points	Number array or string
<u>strokeWidth</u>	Designates the thickness of a line.	An integer or 0 for smallest width on a printed form
<u>style</u>	Allows setting the general appearance of the button. See Button Style for more information.	"regular", "flat", "toolbar", "bevel", "roundedBevel", "gradientBevel", "texturedBevel", "office", "help", "circular", "disclosure", "roundedDisclosure", "custom"
<u>styledText</u>	Enables the possibility of using specific styles in the selected area.	true, false
<u>switchBackWhenReleased</u>	Displays the first picture all the time except when the user clicks the button.	true, false
	Displays the second picture until the mouse button is released.	
<u>switchContinuously</u>	Allows the user to hold down the mouse button to display the pictures continuously (i.e., as an animation).	true, false
<u>switchWhenRollover</u>	Modifies the contents of the picture button when the mouse cursor passes over it. The initial picture is displayed when the cursor leaves the button's area.	true, false
<b>t</b>		
<u>table</u>	Table that the list subform belongs to (if any).	4D table name, or ""
<u>text</u>	The title of the form object	Any text
<u>textAlign</u>	Horizontal location of text within the area that contains it.	"automatic", "right", "center", "justify", "left"
<u>textAngle</u>	Modifies the orientation (rotation) of the text area	0, 90, 180, 270

Property	Description	Possible Values
<a href="#">textDecoration</a>	Sets the selected text to have a line running beneath it.	"normal", "underline"
<a href="#">textFormat</a>	Controls the way the alphanumeric fields and variables appear when displayed or printed.	"#### ####", "(###) ### ####", "### #### #### ####", "### ## ####", "00000", custom formats
<a href="#">textPlacement</a>	Relative location of the button title in relation to the associated icon.	"left", "top", "right", "bottom", "center"
<a href="#">threeState</a>	Allows a check box object to accept a third state.	true, false
<a href="#">timeFormat</a>	Controls the way times appear when displayed or printed. Must only be selected among the 4D built-in formats.	"systemShort", "systemMedium", "systemLong", "iso8601", "hh_mm_ss", "hh_mm", "hh_mm_am", "mm_ss", "HH_MM_SS", "HH_MM", "MM_SS", "blankIfNull" (can be combined with the other possible values)
<a href="#">truncateMode</a>	Controls the display of values when list box columns are too narrow to show their full contents.	"withEllipsis", "none"
<a href="#">type</a>	Mandatory. Designates the data type of the form object.	"text", "rectangle", "groupBox", "tab", "line", "button", "checkbox", "radio", "dropdown", "combo", "webArea", "write", "subform", "plugin", "splitter", "buttonGrid", "progress", "ruler", "spinner", "stepper", "list", "pictureButton", "picturePopup", "listbox", "input", "view"
<a href="#">tooltip</a>	Provide users with additional information about a field.	Additional information to help a user
<a href="#">top</a>	Positions an object at the top (centered).	minimum: 0
<b>u</b>		
<a href="#">urlSource</a>	Designates the URL loaded or being loading by the associated Web area.	A URL.
<a href="#">useLastFrameAsDisabled</a>	Enables setting the last thumbnail as the one to display when the button is	true, false

<b>Property</b>	<b>Description</b>	<b>Possible Values</b>
<a href="#">userInterface</a>	disabled 4D View Pro area interface.	"none" (default), "ribbon", "toolbar"
<b>v</b>		
<a href="#">values</a>	List of default values for an array listbox column	ex: "A","B","42"...
<a href="#">variableCalculation</a>	Allows mathematical calculations to be performed.	"none", "minimum", "maximum", "sum", "count", "average", "standardDeviation", "variance", "sumSquare"
<a href="#">verticalAlign</a>	Vertical location of text within the area that contains it.	"automatic", "top", "middle", "bottom"
<a href="#">verticalLineStroke</a>	Defines the color of the vertical lines in a list box (gray by default).	Any CSS value, "transparent", "automatic"
<a href="#">visibility</a>	Allows hiding the object in the Application environment.	"visible", "hidden", "selectedRows", "unselectedRows"
<b>w</b>		
<a href="#">webEngine</a>	Used to choose between two rendering engines for the Web area, depending on the specifics of the application.	"embedded", "system"
<a href="#">width</a>	Designates an object's horizontal size	minimum: 0
<a href="#">withFormulaBar</a>	Manages the display of a formula bar with the Toolbar interface in the 4D View Pro area.	true, false
<a href="#">wordwrap</a>	Manages the display of contents when it exceeds the width of the object.	"automatic" (excluding list box), "normal", "none"
<b>z</b>		
<a href="#">zoom</a>	Zoom percentage for displaying 4D Write Pro area	number (minimum=0)

## Action

### Draggable

Control whether and how the user can drag the object. By default, no drag operation is allowed.

Two drag modes are available:

- **Custom:** In this mode, any drag operation performed on the object triggers the `On Begin Drag` form event in the context of the object. You then manage the drag action using a method.  
In custom mode, basically the whole drag-and-drop operation is handled by the programmer. This mode lets you implement any interface based upon drag-on-drop, including interfaces that do not necessarily transport data, but can perform any action like opening files or triggering a calculation. This mode is based upon a combination of specific properties, events, and commands from the `Pasteboard` theme.
- **Automatic:** In this mode, 4D **copies** text or pictures directly from the form object. It can then be used in the same 4D area, between two 4D areas, or between 4D and another application. For example, automatic drag (and drop) lets you copy a value between two fields without using programming:



In this mode, the `On Begin Drag` form event is NOT generated. If you want to "force" the use of the custom drag while automatic drag is enabled, hold down the **Alt** (Windows) or **Option** (macOS) key during the action. This option is not available for pictures.

For more information, refer to [Drag and Drop](#) in the *4D Language Reference* manual.

### JSON Grammar

Name	Data Type	Possible Values
draggingText		"none" (default), "custom", "automatic" (excluding list box)

### Objects Supported

[4D Write Pro areas](#) - [Input](#) - [Hierarchical List](#) - [List Box](#) - [Plug-in Area](#)

### See also

[Droppable](#)

### Droppable

Control whether and how the object can be the destination of a drag and drop operation.

Two drop modes are available:

- **Custom:** In this mode, any drop operation performed on the object triggers the

`On Drag Over` and `On Drop` form events in the context of the object. You then manage the drop action using a method.

In custom mode, basically the whole drag-and-drop operation is handled by the programmer. This mode lets you implement any interface based upon drag-on-drop, including interfaces that do not necessarily transport data, but can perform any action like opening files or triggering a calculation. This mode is based upon a combination of specific properties, events, and commands from the `Pasteboard` theme.

- **Automatic:** In this mode, 4D automatically manages — if possible — the insertion of dragged data of the text or picture type that is dropped onto the object (the data are pasted into the object). The `On Drag Over` and `On Drop` form events are NOT generated. On the other hand, the `On After Edit` (during the drop) and `On Data Change` (when the object loses the focus) events are generated.

For more information, refer to [Drag and Drop](#) in the *4D Language Reference* manual.

## JSON Grammar

Name	Data Type	Possible Values
droppingText		"none" (default), "custom", "automatic" (excluding list box)

## Objects Supported

[4D Write Pro areas](#) - [Button](#) - [Input](#) - [Hierarchical List](#) - [List Box](#) - [Plug-in Area](#)

## See also

[Draggable](#)

## Execute object method

When this option is enabled, the object method is executed with the `On Data Change` event *at the same moment* the user changes the value of the indicator. When the option is disabled, the method is executed *after* the modification.

## JSON Grammar

Name	Data Type	Possible Values
continuousExecution	boolean	true, false

## Objects Supported

[Progress bar](#) - [Ruler](#) - [Stepper](#)

## Method

Reference of a method attached to the object. Object methods generally "manage" the object while the form is displayed or printed. You do not call an object method—4D calls it automatically when an event involves the object to which the object method is attached.

Several types of method references are supported:

- a standard object method file path, i.e. that uses the following pattern:

ObjectMethods/objectName.4dm

... where `objectName` is the actual [object name](#). This type of reference indicates that the method file is located at the default location ("sources/forms/formName/ObjectMethods/"). In this case, 4D automatically handles the object method when operations are executed on the form object (renaming, duplication, copy/paste...)

- a project method name: name of an existing project method without file extension, i.e.: `myMethod` In this case, 4D does not provide automatic support for object operations.
- a custom method file path including the .4dm extension, e.g.:  
`.../CustomMethods/myMethod.4dm` You can also use a filesystem:  
`/RESOURCES/Buttons/bOK.4dm` In this case, 4D does not provide automatic support for object operations.

## JSON Grammar

Name	Data Type	Possible Values
methodText		Object method standard or custom file path, or project method name

## Objects Supported

[4D View Pro Area](#) - [4D Write Pro Area](#) - [Button](#) - [Button Grid](#) - [Check Box](#) - [Combo Box](#) - [Dropdown list](#) - [Forms](#) - [Hierarchical List](#) - [Input](#) - [List Box](#) - [List Box Column](#) - [Picture Button](#) - [Picture Pop up menu](#) - [Plug-in Area](#) - [Progress Indicators](#) - [Radio Button](#) - [Ruler](#) - [Spinner](#) - [Splitter](#) - [Stepper](#) - [Subform](#) - [Tab control](#) - [Web Area](#)

## Movable Rows

Array type list boxes

Authorizes the movement of rows during execution. This option is selected by default. It is not available for [selection type list boxes](#) nor for [list boxes in hierarchical mode](#).

## JSON Grammar

Name	Data Type	Possible Values
movableRows	boolean	true, false

## Objects Supported

[List Box](#)

## Multi-selectable

Allows the selection of multiple records/options in a [hierarchical list](#).

## JSON Grammar

Name	Data Type	Possible Values
selectionMode	text	"multiple", "single", "none"

## Objects Supported

[Hierarchical List](#)

## Sortable

Allows sorting column data by clicking a [listbox](#) header. This option is selected by default. Picture type arrays (columns) cannot be sorted using this feature.

In list boxes based on a selection of records, the standard sort function is available only:

- When the data source is *Current Selection*,
- With columns associated with fields (of the Alpha, Number, Date, Time or Boolean type).

In other cases (list boxes based on named selections, columns associated with expressions), the standard sort function is not available. A standard list box sort changes the order of the current selection in the database. However, the highlighted records and the current record are not changed. A standard sort synchronizes all the columns of the list box, including calculated columns.

## JSON Grammar

### Name Data Type Possible Values

sortable boolean true, false

## Objects Supported

[List Box](#)

## Standard action

Typical activities to be performed by active objects (e.g., letting the user accept, cancel, or delete records, move between records or from page to page in a multi-page form, etc.) have been predefined by 4D as standard actions. They are described in detail in the [Standard actions](#) section of the *Design Reference*.

You can assign both a standard action and a project method to an object. In this case, the standard action is usually executed after the method and 4D uses this action to enable/disable the object according to the current context. When an object is deactivated, the associated project method cannot be executed.

You can also set this property using the `OBJECT SET ACTION` command.

## JSON Grammar

Name	Data Type	Possible Values
action	string	The name of a <a href="#">valid standard action</a> .

## Objects Supported

[Button](#) - [Button Grid](#) - [Check Box](#) - [Drop-down List](#) - [List Box](#) - [Picture Button](#) - [Picture Pop-up Menu](#) - [Tab control](#)



## Animation

### Loop back to first frame

Pictures are displayed in a continuous loop. When the user reaches the last picture and clicks again, the first picture appears, and so forth.

#### JSON Grammar

Name	Data Type	Possible Values
loopBackToFirstFrame	boolean	true, false

#### Objects Supported

[Picture Button](#)

### Switch back when released

Displays the first picture all the time except when the user clicks the button. Displays the second picture until the mouse button is released. This mode allows you to create an action button with a different picture for each state (idle and clicked). You can use this mode to create a 3D effect or display any picture that depicts the action of the button.

#### JSON Grammar

Name	Data Type	Possible Values
switchBackWhenReleased	boolean	true, false

#### Objects Supported

[Picture Button](#)

### Switch continuously on clicks

Allows the user to hold down the mouse button to display the pictures continuously (i.e., as an animation). When the user reaches the last picture, the object does not cycle back to the first picture.

#### JSON Grammar

Name	Data Type	Possible Values
switchContinuously	boolean	true, false

#### Objects Supported

[Picture Button](#)

### Switch every x ticks

Enables cycling through the contents of the picture button at the specified speed (in ticks). In this mode, all other options are ignored.

## **JSON Grammar**

<b>Name</b>	<b>Data Type</b>	<b>Possible Values</b>
frameDelay	integer	minimum: 0

## **Objects Supported**

[Picture Button](#)

## **Switch when roll over**

Modifies the contents of the picture button when the mouse cursor passes over it. The initial picture is displayed when the cursor leaves the button's area.

## **JSON Grammar**

<b>Name</b>	<b>Data Type</b>	<b>Possible Values</b>
switchWhenRollover	boolean	true, false

## **Objects Supported**

[Picture Button](#)

## **Use Last frame as disabled**

Enables setting the last thumbnail as the one to display when the button is disabled. The thumbnail used when the button is disabled is processed separately by 4D: when you combine this option with "Switch Continuously" and "Loop Back to First Frame", the last picture is excluded from the sequence associated with the button and only appears when it is disabled.

## **JSON Grammar**

<b>Name</b>	<b>Data Type</b>	<b>Possible Values</b>
useLastFrameAsDisabled	boolean	true, false

## **Objects Supported**

[Picture Button](#)

# Appearance

## Default Button

The default button property designates the button that gets the initial focus at runtime when no button of the form has the [Focusable](#) property.

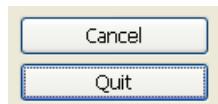
There can only be one default button per form page.

In addition, on macOS, the default button property modifies the button's appearance in order to indicate a "recommended choice" to the user. The default button can be different from the focused button. Default buttons have a specific blue appearance on macOS:



Button must have a standard height to get the default button appearance.

On Windows, the concept of "recommended choice" is not supported: only the focused button has a different appearance at runtime. However, in the 4D form editor, the default button is represented with a blue outline:



## JSON Grammar

Name	Data Type	Possible Values
defaultButton	boolean	true, false

## Objects Supported

[Regular Button](#) - [Flat Button](#)

## Hide focus rectangle

During execution, a field or any enterable area is outlined by a selection rectangle when it has the focus (via the Tab key or a single click). You can hide this rectangle by enabling this property. Hiding the focus rectangle may be useful in the case of specific interfaces.

## JSON Grammar

Name	Data Type	Possible Values
hideFocusRing	boolean	true, false

## Objects Supported

[4D Write Pro area](#) - [Hierarchical List](#) - [Input](#) - [List Box](#) - [Subform](#)

## Hide selection highlight

Selection type list boxes

This property is used to disable the selection highlight in list boxes.

When this option is enabled, the selection highlight is no longer visible for selections made in list boxes. Selections themselves are still valid and work in exactly the same way as previously; however, they are no longer represented graphically onscreen, and you will need to [define their appearance programmatically](#).

By default, this option is not enabled.

## JSON Grammar

Name	Data Type	Possible Values
hideSystemHighlight	boolean	true, false

## Objects Supported

[List Box](#)

## Horizontal Scroll Bar

An interface tool allowing the user to move the viewing area to the left or right.

Available values:

Property List	JSON value	Description
Yes	"visible"	The scrollbar is always visible, even when it is not necessary (in other words, when the size of the object contents is smaller than that of the frame).
No	"hidden"	The scrollbar is never visible
Automatic	"automatic"	The scrollbar appears automatically whenever necessary and the user can enter text larger than the object width

Picture objects can have scrollbars when the display format of the picture is set to "Truncated (non-centered)."

## JSON Grammar

Name	Data Type	Possible Values
scrollbarHorizontal	text	"visible", "hidden", "automatic"

## Objects Supported

[Hierarchical List](#) - [Subform](#) - [List Box](#) - [Input](#) - [4D Write Pro area](#)

## See also

[Vertical scroll bar](#)

## Resolution

Sets the screen resolution for the 4D Write Pro area contents. By default, it is set to 72 dpi (macOS), which is the standard resolution for 4D forms on all platforms. Setting this property to 96 dpi will set a windows/web rendering on both macOS and Windows platforms. Setting this property to **automatic** means that document rendering will differ between macOS and Windows platforms.

## JSON Grammar

Name	Data Type	Possible Values
dpi	number	0=automatic, 72, 96

## Objects Supported

[4D Write Pro area](#)

## Show background

Displays/hides both background images and background color.

## JSON Grammar

Name	Data Type	Possible Values
showBackground	boolean	true (default), false

## Objects Supported

[4D Write Pro area](#)

## Show footers

Displays/hides the footers when [Page view mode](#) is set to "Page".

## JSON Grammar

Name	Data Type	Possible Values
showFooters	boolean	true (default), false

## Objects Supported

[4D Write Pro area](#)

## Show Formula Bar

When enabled, the formula bar is visible below the Toolbar interface in the 4D View Pro area. If not selected, the formula bar is hidden.

This property is available only for the [Toolbar](#) interface.

## JSON Grammar

Name	Data Type	Possible Values
withFormulaBar	boolean	true (default), false

## Objects Supported

[4D View Pro area](#)

## Show headers

Displays/hides the headers when [Page view mode](#) is set to "Page".

### JSON Grammar

Name	Data Type	Possible Values
showHeaders	boolean	true (default), false

### Objects Supported

[4D Write Pro area](#)

## Show hidden characters

Displays/hides invisible characters

### JSON Grammar

Name	Data Type	Possible Values
showHiddenChars	boolean	true (default), false

### Objects Supported

[4D Write Pro area](#)

## Show horizontal ruler

Displays/hides the horizontal ruler when the document view is in [Page mode](#).

### JSON Grammar

Name	Data Type	Possible Values
showHorizontalRuler	boolean	true (default), false

### Objects Supported

[4D Write Pro area](#)

## Show HTML WYSIWIG

Enables/disables the HTML WYSIWYG view, in which any 4D Write Pro advanced attributes which are not compliant with all browsers are removed.

### JSON Grammar

Name	Data Type	Possible Values
showHTMLWysiwyg	boolean	true, false (default)

### Objects Supported

[4D Write Pro area](#)

## Show page frame

Displays/hides the page frame when [Page view mode](#) is set to "Page".

## JSON Grammar

Name	Data Type	Possible Values
showPageFrames	boolean	true, false

## Objects Supported

[4D Write Pro area](#)

## Show references

Displays all 4D expressions inserted in the 4D Write Pro document as *references*. When this option is disabled, 4D expressions are displayed as *values*. By default when you insert a 4D field or expression, 4D Write Pro computes and displays its current value. Select this property if you wish to know which field or expression is displayed. The field or expression references then appear in your document, with a gray background.

For example, you have inserted the current date along with a format, the date is displayed:

July 11, 2016

With the Show references property on, the reference is displayed:

String(Current date;Internal date long)

4D expressions can be inserted using the `ST INSERT EXPRESSION` command.

## JSON Grammar

Name	Data Type	Possible Values
showReferences	boolean	true, false (default)

## Objects Supported

[4D Write Pro area](#)

## Show vertical ruler

Displays/hides the vertical ruler when the document view is in [Page mode](#).

## JSON Grammar

Name	Data Type	Possible Values
showVerticalRuler	boolean	true (default), false

## Objects Supported

[4D Write Pro area](#)

## Tab Control Direction

You can set the direction of tab controls in your forms. This property is available on all the platforms but can only be displayed in macOS. You can choose to place the tab controls on top (standard) or on the bottom.

When tab controls with a custom direction are displayed under Windows, they automatically return to the standard direction (top).

## JSON Grammar

Name	Data Type	Possible Values
labelsPlacement	boolean	"top", "bottom"

## Objects Supported

[Tab Control](#)

## User Interface

You can add an interface to 4D View Pro areas to allow end users to perform basic modifications and data manipulations. 4D View Pro offers two optional interfaces to choose from, **Ribbon** and **Toolbar**.

## JSON Grammar

Name	Data Type	Possible Values
userInterface	text	"none" (default), "ribbon", "toolbar"

## Objects Supported

[4D View Pro area](#)

## See also

[4D View Pro reference guide](#)

## Vertical Scroll Bar

An interface tool allowing the user to move the viewing area up and down.

Available values:

Property List	JSON value	Description
Yes	"visible"	The scrollbar is always visible, even when it is not necessary (in other words, when the size of the object contents is smaller than that of the frame).
No	"hidden"	The scrollbar is never visible
Automatic	"automatic"	The scrollbar appears automatically whenever necessary (in other words, when the size of the object contents is greater than that of the frame)

Picture objects can have scrollbars when the display format of the picture is set to "Truncated (non-centered)."

If a text input object does not have a scroll bar, the user can scroll the information using the arrow keys.

## JSON Grammar

Name	Data Type	Possible Values
scrollbarVerticaltext		"visible", "hidden", "automatic"

## Objects Supported

[Hierarchical List](#) - [Subform](#) - [List Box](#) - [Input](#) - [4D Write Pro area](#)

## See also

[Horizontal scroll bar](#)

## View mode

Sets the mode for displaying the 4D Write Pro document in the form area. Three values are available:

- **Page**: the most complete view mode, which includes page outlines, orientation, margins, page breaks, headers and footers, etc.
- **Draft**: draft mode with basic document properties
- **Embedded**: view mode suitable for embedded areas; it does not display margins, footers, headers, page frames, etc. This mode can also be used to produce a web-like view output (if you also select the [96 dpi resolution](#) and the [Show HTML WYSIWYG](#) properties).

The View mode property is only used for onscreen rendering. Regarding printing settings, specific rendering rules are automatically used.

## JSON Grammar

Name	Data Type	Possible Values
layoutModetext		"page", "draft", "embedded"

## Objects Supported

[4D Write Pro area](#)

## Zoom

Sets the zoom percentage for displaying 4D Write Pro area contents.

## JSON Grammar

Name	Data Type	Possible Values
zoom	number	minimum = 0

## Objects Supported

[4D Write Pro area](#)

# Background and Border

## Alternate Background Color

Allows setting a different background color for odd-numbered rows/columns in a list box. By default, *Automatic* is selected: the column uses the alternate background color set at the list box level.

### JSON Grammar

Name	Data Type	Possible Values
alternateFill	string	any css value; "transparent"; "automatic"; "automaticAlternate"

### Objects Supported

[List Box](#) - [List Box Column](#)

## Background Color / Fill Color

Defines the background color of an object.

In the case of a list box, by default *Automatic* is selected: the column uses the background color set at the list box level.

### JSON Grammar

Name	Data Type	Possible Values
fill	string	any css value; "transparent"; "automatic"

### Objects Supported

[Hierarchical List](#) - [List Box](#) - [List Box Column](#) - [List Box Footer](#) - [Oval](#) - [Rectangle](#) - [Text Area](#)

### See also

[Transparent](#)

## Background Color Expression

Selection and collection type list boxes

An expression or a variable (array variables cannot be used) to apply a custom background color to each row of the list box. The expression or variable will be evaluated for each row displayed and must return a RGB color value. For more information, refer to the description of the `OBJECT SET RGB COLORS` command in the *4D Language Reference manual*.

You can also set this property using the `LISTBOX SET PROPERTY` command with `1k background color expression` constant.

With collection or entity selection type list boxes, this property can also be

set using a [Meta Info Expression](#).

## JSON Grammar

Name	Data Type	Possible Values
rowFillSource	string	An expression returning a RGB color value

## Objects Supported

[List Box](#) - [List Box Column](#)

## Border Line Style

Allows setting a standard style for the object border.

## JSON Grammar

Name	Data Type	Possible Values
borderStyle	text	"system", "none", "solid", "dotted", "raised", "sunken", "double"

## Objects Supported

[4D View Pro Area](#) - [4D Write Pro areas](#) - [Buttons](#) - [Button Grid](#) - [Hierarchical List](#) - [Input](#) - [List Box](#) - [Picture Button](#) - [Picture Pop-up Menu](#) - [Plug-in Area](#) - [Progress Indicator](#) - [Ruler](#) - [Spinner](#) - [Stepper](#) - [Subform](#) - [Text Area](#) - [Web Area](#)

## Dotted Line Type

Describes dotted line type as a sequence of black and white points.

## JSON Grammar

Name	Data Type	Possible Values
strokeDashArray	number array or string	Ex. "6 1" or [6,1] for a sequence of 6 black point and 1 white point

## Objects Supported

[Rectangle](#) - [Oval](#) - [Line](#)

## Hide extra blank rows

Controls the display of extra blank rows added at the bottom of a list box object. By default, 4D adds such extra rows to fill the empty area:

Days	Values
Monday	5
Tuesday	6
Wednesday	41
Thursday	66
Friday	12
Saturday	2
Sunday	88

Extra blank rows

You can remove these empty rows by selecting this option. The bottom of the list box object is then left blank:

Days	Values
Monday	5
Tuesday	6
Wednesday	41
Thursday	66
Friday	12
Saturday	2
Sunday	88

## JSON Grammar

Name	Data Type	Possible Values
hideExtraBlankRows	boolean	true, false

## Objects Supported

### [List Box](#)

## Line Color

Designates the color of the object's lines. The color can be specified by:

- a color name - like "red"
- a HEX value - like "#ff0000"
- an RGB value - like "rgb(255,0,0)"

You can also set this property using the [OBJECT SET RGB COLORS](#) command.

## JSON Grammar

Name	Data Type	Possible Values
stroke	string	any css value, "transparent", "automatic"

This property is also available for text based objects, in which case it designates both the font color and the object's lines, see [Font color](#).

## Objects Supported

[Line](#) - [Oval](#) - [Rectangle](#)

## Line Width

Designates the thickness of a line.

### JSON Grammar

Name	Data Type	Possible Values
strokeWidth	number	0 for smallest width on a printed form, or any integer value < 20

## Objects Supported

[Line](#) - [Oval](#) - [Rectangle](#)

## Row Background Color Array

Array type list boxes

The name of an array to apply a custom background color to each row of the list box or column.

The name of a Longint array must be entered. Each element of this array corresponds to a row of the list box (if applied to the list box) or to a cell of the column (if applied to a column), so the array must be the same size as the array associated with the column. You can use the constants of the [SET RGB COLORS](#) theme. If you want the cell to inherit the background color defined at the higher level, pass the value -255 to the corresponding array element.

For example, given a list box where the rows have an alternating gray/light gray color, defined in the properties of the list box. A background color array has also been set for the list box in order to switch the color of rows where at least one value is negative to light orange:

```
<>_BgndColors{$i}:=0x00FFD0B0 // orange  
<>_BgndColors{$i}:=-255 // default value
```

Next you want to color the cells with negative values in dark orange. To do this, you set a background color array for each column, for example `<>_BgndColor_1`, `<>_BgndColor_2` and `<>_BgndColor_3`. The values of these arrays have priority over the ones set in the list box properties as well as those of the general background color array:

```
<>_BgndColorsCol_3{2}:=0x00FF8000 // dark orange  
<>_BgndColorsCol_2{5}:=0x00FF8000  
<>_BgndColorsCol_1{9}:=0x00FF8000  
<>_BgndColorsCol_1{16}:=0x00FF8000
```

You can get the same result using the `LISTBOX SET ROW FONT STYLE` and `LISTBOX SET ROW COLOR` commands. They have the advantage of letting you skip having to predefine style/color arrays for the columns: instead they are created dynamically by

the commands.

## JSON Grammar

Name	Data Type	Possible Values
rowFillSource	string	The name of a longint array.

## Objects Supported

[List Box](#) - [List Box Column](#)

## Transparent

Sets the list box background to "Transparent". When set, any [alternate background color](#) or [background color](#) defined for the column is ignored.

## JSON Grammar

Name	Data Type	Possible Values
fill	text	"transparent"

## Objects Supported

[List Box](#)

## See also

[Background Color / Fill Color](#)

## Coordinates & Sizing

### Automatic Row Height

This property is only available for list boxes with the following [data sources](#):

- collection or entity selection,
- array (non-hierarchical).

The property is not selected by default. When used for at least one column, the height of every row in the column will automatically be calculated by 4D, and the column contents will be taken into account. Note that only columns with the option selected will be taken into account to calculate the row height.

#### NOTE

When resizing the form, if the "Grow" [horizontal sizing](#) property was assigned to the list box, the right-most column will be increased beyond its maximum width if necessary.

When this property is enabled, the height of every row is automatically calculated in order to make the cell contents entirely fit without being truncated (unless the [Wordwrap](#) option is disabled).

- The row height calculation takes into account:
  - any content types (text, numerics, dates, times, pictures (calculation depends on the picture format), objects),
  - any control types (inputs, check boxes, lists, dropdowns),
  - fonts, font styles and font sizes,
  - the [Wordwrap](#) option: if disabled, the height is based on the number of paragraphs (lines are truncated); if enabled, the height is based on number of lines (not truncated).
- The row height calculation ignores:
  - hidden column contents
  - [Row Height](#) and [Row Height Array](#) properties (if any) set either in the Property list or by programming.

#### CAUTION

Since it requires additional calculations at runtime, the automatic row height option could affect the scrolling fluidity of your list box, in particular when it contains a large number of rows.

### JSON Grammar

Name	Data Type	Possible Values
rowHeightAuto	boolean	true, false

### Objects Supported

[List Box Column](#)

## Bottom

Bottom coordinate of the object in the form.

### JSON Grammar

#### Name Data Type Possible Values

bottom number minimum: 0

### Objects Supported

[4D View Pro Area](#) - [4D Write Pro Area](#) - [Button](#) - [Button Grid](#) - [Check Box](#) - [Combo Box](#) - [Dropdown list](#) - [Group Box](#) - [Hierarchical List](#) - [Input](#) - [List Box](#) - [Line](#) - [List Box Column](#) - [Oval](#) - [Picture Button](#) - [Picture Pop up menu](#) - [Plug-in Area](#) - [Progress Indicators](#) - [Radio Button](#) - [Rectangle](#) - [Ruler](#) - [Spinner](#) - [Splitter](#) - [Static Picture](#) - [Stepper](#) - [Subform](#) - [Tab control](#) - [Text Area](#) - [Web Area](#)

## Left

Left coordinate of the object on the form.

### JSON Grammar

#### Name Data Type Possible Values

left number minimum: 0

### Objects Supported

[4D View Pro Area](#) - [4D Write Pro Area](#) - [Button](#) - [Button Grid](#) - [Check Box](#) - [Combo Box](#) - [Dropdown list](#) - [Group Box](#) - [Hierarchical List](#) - [Input](#) - [List Box](#) - [Line](#) - [List Box Column](#) - [Oval](#) - [Picture Button](#) - [Picture Pop up menu](#) - [Plug-in Area](#) - [Progress Indicators](#) - [Radio Button](#) - [Ruler](#) - [Rectangle](#) - [Spinner](#) - [Splitter](#) - [Static Picture](#) - [Stepper](#) - [Subform](#) - [Tab control](#) - [Text Area](#) - [Web Area](#)

## Right

Right coordinate of the object in the form.

### JSON Grammar

#### Name Data Type Possible Values

right number minimum: 0

### Objects Supported

[4D View Pro Area](#) - [4D Write Pro Area](#) - [Button](#) - [Button Grid](#) - [Check Box](#) - [Combo Box](#) - [Dropdown list](#) - [Group Box](#) - [Hierarchical List](#) - [Input](#) - [List Box](#) - [Line](#) - [List Box Column](#) - [Oval](#) - [Picture Button](#) - [Picture Pop up menu](#) - [Plug-in Area](#) - [Progress Indicators](#) - [Radio Button](#) - [Ruler](#) - [Rectangle](#) - [Spinner](#) - [Splitter](#) - [Static Picture](#) - [Stepper](#) - [Subform](#) - [Tab control](#) - [Text Area](#) - [Web Area](#)

## Top

Top coordinate of the object in the form.

### JSON Grammar

## Name Data Type Possible Values

top number minimum: 0

## Objects Supported

[4D View Pro Area](#) - [4D Write Pro Area](#) - [Button](#) - [Button Grid](#) - [Check Box](#) - [Combo Box](#) - [Dropdown list](#) - [Group Box](#) - [Hierarchical List](#) - [Input](#) - [List Box](#) - [Line](#) - [List Box Column](#) - [Oval](#) - [Picture Button](#) - [Picture Pop up menu](#) - [Plug-in Area](#) - [Progress Indicators](#) - [Radio Button](#) - [Ruler](#) - [Rectangle](#) - [Spinner](#) - [Splitter](#) - [Static Picture](#) - [Stepper](#) - [Subform](#) - [Tab control](#) - [Text Area](#) - [Web Area](#)

## Corner Radius

### ▪ History

Defines the corner roundness (in pixels) of the object. By default, the radius value is 0 pixels. You can change this property to draw rounded objects with custom shapes:

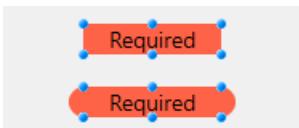


Minimum value is 0, in this case a standard non-rounded object rectangle is drawn. Maximum value depends on the rectangle size (it cannot exceed half the size of the shortest rectangle side) and is calculated dynamically.

### ⓘ NOTE

With [text areas](#) and [inputs](#):

- the corner radius property is only available with "none", "solid", or "dotted" [border line styles](#),
- the corner roundness is drawn outside the area of the object (the object appears larger in the form but its [width](#) and [height](#) are not extended).



You can also set this property using the [OBJECT Get corner radius](#) and [OBJECT SET CORNER RADIUS](#) commands.

## JSON Grammar

### Name Data Type Possible Values

borderRadius integer minimum: 0

## Objects Supported

[Input](#) - [Rectangle](#) - [Text Area](#)

## Height

This property designates an object's vertical size.

Some objects may have a predefined height that cannot be altered.

## JSON Grammar

### Name Data Type Possible Values

height number minimum: 0

### Objects Supported

[4D View Pro Area](#) - [4D Write Pro Area](#) - [Button](#) - [Button Grid](#) - [Check Box](#) - [Combo Box](#) - [Dropdown list](#) - [Group Box](#) - [Hierarchical List](#) - [Input](#) - [List Box](#) - [Line](#) - [List Box Column](#) - [Oval](#) - [Picture Button](#) - [Picture Pop up menu](#) - [Plug-in Area](#) - [Progress Indicators](#) - [Radio Button](#) - [Ruler](#) - [Rectangle](#) - [Spinner](#) - [Splitter](#) - [Static Picture](#) - [Stepper](#) - [Subform](#) - [Tab control](#) - [Text Area](#) - [Web Area](#)

## Width

This property designates an object's horizontal size.

- Some objects may have a predefined height that cannot be altered.
- If the [Resizable](#) property is used for a [list box column](#), the user can also manually resize the column.
- When resizing the form, if the ["Grow" horizontal sizing](#) property was assigned to the list box, the right-most column will be increased beyond its maximum width if necessary.

## JSON Grammar

### Name Data Type Possible Values

width number minimum: 0

### Objects Supported

[4D View Pro Area](#) - [4D Write Pro Area](#) - [Button](#) - [Button Grid](#) - [Check Box](#) - [Combo Box](#) - [Dropdown list](#) - [Group Box](#) - [Hierarchical List](#) - [Input](#) - [Line](#) - [List Box](#) - [List Box Column](#) - [Oval](#) - [Picture Button](#) - [Picture Pop up menu](#) - [Plug-in Area](#) - [Progress Indicators](#) - [Radio Button](#) - [Ruler](#) - [Rectangle](#) - [Spinner](#) - [Splitter](#) - [Static Picture](#) - [Stepper](#) - [Subform](#) - [Tab control](#) - [Text Area](#) - [Web Area](#)

## Maximum Width

The maximum width of the column (in pixels). The width of the column cannot be increased beyond this value when resizing the column or form.

When resizing the form, if the ["Grow" horizontal sizing](#) property was assigned to the list box, the right-most column will be increased beyond its maximum width if necessary.

## JSON Grammar

### Name Data Type Possible Values

maxWidth number minimum: 0

### Objects Supported

[List Box Column](#)

## Minimum Width

The minimum width of the column (in pixels). The width of the column cannot be reduced below this value when resizing the column or form.

When resizing the form, if the "["Grow" horizontal sizing](#)" property was assigned to the list box, the right-most column will be increased beyond its maximum width if necessary.

## JSON Grammar

Name	Data Type	Possible Values
minWidth	number	minimum: 0

## Objects Supported

[List Box Column](#)

## Row Height

Sets the height of list box rows (excluding headers and footers). By default, the row height is set according to the platform and the font size.

## JSON Grammar

Name	Data Type	Possible Values
rowHeight	string	css value in unit "em" or "px" (default)

## Objects Supported

[List Box](#)

## See also

[Row Height Array](#)

## Row Height Array

This property is used to specify the name of a row height array that you want to associate with the list box. A row height array must be of the numeric type (longint by default).

When a row height array is defined, each of its elements whose value is different from 0 (zero) is taken into account to determine the height of the corresponding row in the list box, based on the current Row Height unit.

For example, you can write:

```
ARRAY LONGINT (RowHeights;20)
RowHeights{5} :=3
```

Assuming that the unit of the rows is "lines," then the fifth row of the list box will have a height of three lines, while every other row will keep its default height.

- The Row Height Array property is not taken into account for hierarchical list boxes.
- For array and collection/entity selection list boxes, this property is available only if the [Automatic Row Height](#) option is not selected.

## JSON Grammar

Name	Data Type	Possible Values
rowHeightSource	string	Name of a 4D array variable.

## Objects Supported

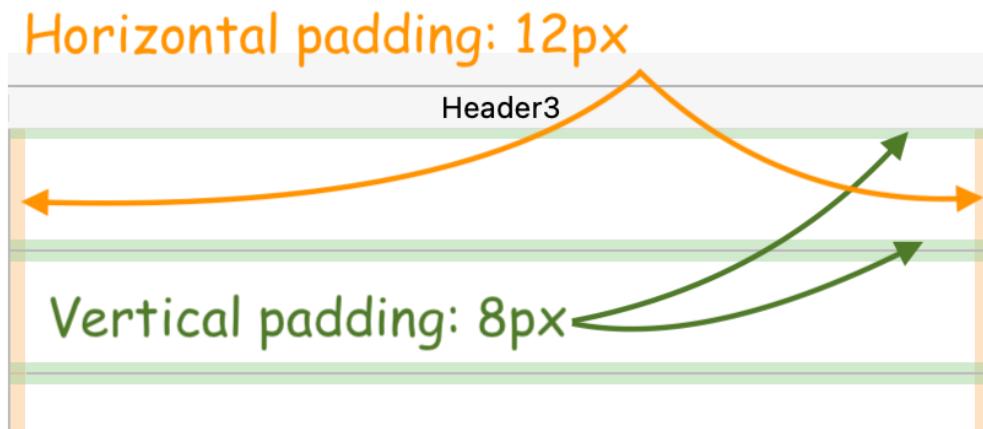
[List Box](#)

## See also

[Row Height](#)

## Horizontal Padding

Sets a horizontal padding for the cells. The value is set in pixels (default = 0).



## JSON Grammar

Name	Data Type	Possible Values
horizontalPadding	number	Number of pixels (must be $\geq 0$ )

## Objects Supported

[List Box](#) - [List Box Column](#) - [Footers](#) - [Headers](#)

## See also

[Vertical Padding](#)

## Vertical Padding

Sets a vertical padding for the cells. The value is set in pixels (default = 0).

## JSON Grammar

Name	Data Type	Possible Values
verticalPadding	number	Number of pixels (must be $\geq 0$ )

## **Objects Supported**

[List Box](#) - [List Box Column](#) - [Footers](#) - [Headers](#)

## **See also**

[Horizontal Padding](#)

## Crop

### Columns

Sets the number of columns in a thumbnail table.

#### JSON Grammar

**Name      Data Type Possible Values**

columnCount integer minimum: 1

#### Objects Supported

[Picture Button](#) - [Button Grid](#) - [Picture Pop-up Menu](#)

## Rows

Sets the number of rows in a thumbnail table.

#### JSON Grammar

**Name      Data Type Possible Values**

rowCount integer minimum: 1

#### Objects Supported

[Picture Button](#) - [Button Grid](#) - [Picture Pop-up Menu](#)

## Data Source

### Automatic Insertion

When this option is selected, if a user enters a value that is not found in the list associated with the object, this value is automatically added to the list stored in memory.

When the **automatic insertion** option is not set (default), the value entered is stored in the form object but not in the list in memory.

This property is supported by:

- [Combo box](#) and [list box column](#) form objects associated to a choice list.
- [Combo box](#) form objects whose associated list is filled by their array or object datasource.

For example, given a choice list containing "France, Germany, Italy" that is associated with a "Countries" combo box: if the **automatic insertion** property is set and a user enters "Spain", then the value "Spain" is automatically added to the list in memory:



If the choice list was created from a list defined in Design mode, the original list is not modified.

### JSON Grammar

Name	Data Type	Possible Values
automaticInsertion	boolean	true, false

### Objects Supported

[Combo Box - List Box Column](#)

## Choice List

Associates a choice list with an object. It can be a choice list name (a list reference) or a collection of default values.

You can also associate choice lists to objects using the [OBJECT SET LIST BY NAME](#) or [OBJECT SET LIST BY REFERENCE](#) commands.

### JSON Grammar

Name	Data Type	Possible Values
choiceList	list, collection	A list of possible values
list	list, collection	A list of possible values (hierarchical lists only)

### Objects Supported

## Choice List (static list)

List of static values to use as labels for the tab control object.

### JSON Grammar

Name	Data Type	Possible Values
labels	list, collection	A list of values to fill the tab control

### Objects Supported

[Tab Control](#)

## Current item

Collection or entity selection list boxes

Specifies a variable or expression that will be assigned the collection element/entity selected by the user. You must use an object variable or an assignable expression that accepts objects. If the user does not select anything or if you used a collection of scalar values, the Null value is assigned.

This property is "read-only", it is automatically updated according to user actions in the list box. You cannot edit its value to modify the list box selection status.

### JSON Grammar

Name	Data Type	Possible Values
currentItemSource	string	Object expression

### Objects Supported

[List Box](#)

## Current item position

Collection or entity selection list boxes

Specifies a variable or expression that will be assigned a longint indicating the position of the collection element/entity selected by the user.

- if no element/entity is selected, the variable or expression receives zero,
- if a single element/entity is selected, the variable or expression receives its location,
- if multiple elements/entities are selected, the variable or expression receives the position of element/entity that was last selected.

This property is "read-only", it is automatically updated according to user actions in the list box. You cannot edit its value to modify the list box selection status.

### JSON Grammar

Name	Data Type	Possible Values
currentItemPositionSource	string	Number expression

## Objects Supported

[List Box](#)

## Data Type (expression type)

Defines the data type for the displayed expression. This property is used with:

- [List box columns](#) of the selection and collection types.
- [Drop-down lists](#) associated to objects or arrays.

See also [Expression Type](#) section.

## JSON Grammar

Name	Data Type	Possible Values
dataSourceTypeHint	string	<ul style="list-style-type: none"> <li>• <b>list box columns:</b> "boolean", "number", "picture", "text", "date", "time". <i>Array/selection list box only:</i> "integer", "object"</li> <li>• <b>drop-down lists:</b> "object", "arrayText", "arrayDate", "arrayTime", "arrayNumber"</li> </ul>

## Objects Supported

[Drop-down Lists](#) associated to objects or arrays - [List Box column](#)

## Data Type (list)

Defines the type of data to save in the field or variable associated to the [drop-down list](#). This property is used with:

- Drop-down lists [associated to a choice list](#).
- Drop-down lists [associated to a hierarchical choice list](#).

Three options are available:

- **List reference:** declares that the drop-down list is hierarchical. It means that the drop-down list can display up to two hierarchical levels and its contents can be managed by the 4D language commands of the **Hierarchical Lists** theme.
- **Selected item value** (default): the drop-down list is not hierarchical and the value of the item chosen in the list by the user is saved directly. For example, if the user chooses the value "Blue", then this value is saved in the field.
- **Selected item reference:** the drop-down list is not hierarchical and the reference of the choice list item is saved in the object. This reference is the numeric value associated with each item either through the *itemRef* parameter of the [APPEND TO LIST](#) or [SET LIST ITEM](#) commands, or in the list editor. This option lets you optimize memory usage: storing numeric values in fields uses less space than storing strings. It also makes it easier to translate applications: you just create multiple lists in different languages but with the same item references, then load the list based on the language of the application.

Using the **Selected item reference** option requires compliance with the following

principles:

- To be able to store the reference, the field or variable data source must be of the Number type (regardless of the type of value displayed in the list). The [expression](#) property is automatically set.
- Valid and unique references must be associated with list items.
- The drop-down list must be associated with a field or a variable.

## JSON Grammar

### Name Data Type Possible Values

saveAsstring "value", "reference"

Setting only `"dataSourceTypeHint" : "integer"` with a `"type": "dropdown"` form object will declare a hierarchical drop-down list.

## Objects Supported

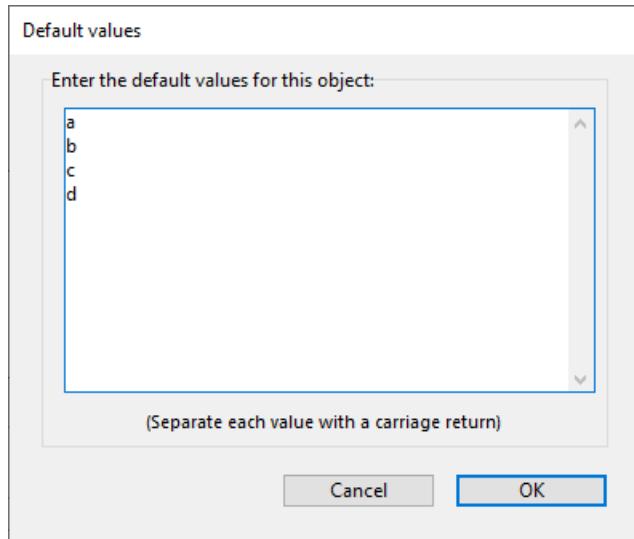
[Drop-down Lists](#) associated to lists

## Default (list of) values

List of values that will be used as default values for the list box column (array type only). These values are automatically available in the [array variable](#) associated with this column when the form is executed. Using the language, you can manage the object by referring to this array.

Do not make confusion between this property and the "[default value](#)" property that allows to define a field value in new records.

You must enter a list of values. In the Form editor, a specific dialog box allows you to enter values separated by carriage returns:



You can also define a [choice list](#) with the list box column. However, a choice list will be used as list of selectable values for each column row, whereas the default list fill all column rows.

## JSON Grammar

Name	Data Type	Possible Values
values collection		A collection of default values (strings), ex: "a", "b", "c", "d"

## Objects Supported

[List Box Column \(array type only\)](#)

## Expression

This description is specific to [selection](#) and [collection](#) type list box columns. See also [Variable or Expression](#) section.

A 4D expression to be associated with a column. You can enter:

- A **simple variable** (in this case, it must be explicitly declared for compilation). You can use any type of variable except BLOBS and arrays. The value of the variable will be generally calculated in the `On Display Detail` event.
- A **field** using the standard [Table]Field syntax ([selection type list box](#) only), for example: `[Employees] LastName`. The following types of fields can be used:
  - String
  - Numeric
  - Date
  - Time
  - Picture
  - Boolean
 You can use fields from the Master Table or from other tables.

- A **4D expression** (simple expression, formula or 4D method). The expression must return a value. The value will be evaluated in the `On Display Detail` and `On Data Change` events. The result of the expression will be automatically displayed when you switch to Application mode. The expression will be evaluated for each record of the selection (current or named) of the Master Table (for selection type list boxes), each element of the collection (for collection type list boxes) or each entity of the selection (for entity selection list boxes). If it is empty, the column will not display any results. The following expression types are supported:
  - String
  - Numeric
  - Date
  - Picture
  - Boolean

For collection/entity selection list boxes, Null or unsupported types are displayed as empty strings.

When using collections or entity selections, you will usually declare the element property or entity attribute associated to a column within an expression containing `This`. `This` is a dedicated 4D command that returns a reference to the currently processed element. For example, you can use `This.\<propertyPath>` where `\<propertyPath>` is the path of a property in the collection or an entity attribute path to access the current value of

each element/entity. If you use a collection of scalar values, 4D will create an object for each collection element with a single property (named "value"), filled with the element value. In this case, you will use `This.value` as expression.

If a [non-assignable expression](#) is used (e.g. `[Person]FirstName + [Person]LastName`), the column is never enterable even if the [Enterable](#) property is enabled.

If a field, a variable, or an assignable expression (e.g. `Person.lastName`) is used, the column can be enterable or not depending on the [Enterable](#) property.

## JSON Grammar

Name	Data Type	Possible Values
dataSource	string	A 4D variable, field name, or an arbitrary complex language expression.

## Objects Supported

### [List Box Column](#)

## Master Table

Current selection list boxes

Specifies the table whose current selection will be used. This table and its current selection will form the reference for the fields associated with the columns of the list box (field references or expressions containing fields). Even if some columns contain fields from other tables, the number of rows displayed will be defined by the master table.

All database tables can be used, regardless of whether the form is related to a table (table form) or not (project form).

## JSON Grammar

### Name Data Type Possible Values

table number Table number

## Objects Supported

### [List Box](#)

## Save as

This property is available in the following conditions:

- a [choice list](#) is associated with the object
- for [inputs](#) and [list box columns](#), a [required list](#) is also defined for the object (both options should use usually the same list), so that only values from the list can be entered by the user.

This property specifies, in the context of a field or variable associated with a list of values, the type of contents to save:

- **Save as Value** (default option): the value of the item chosen in the list by the user is saved directly. For example, if the user chooses the value "Blue", then this value is saved in the field.
- **Save as Reference**: the reference of the choice list item is saved in the object. This reference is the numeric value associated with each item either through the *itemRef* parameter of the [APPEND TO LIST](#) or [SET LIST ITEM](#) commands, or in the list editor.

This option lets you optimize memory usage: storing numeric values in fields uses less space than storing strings. It also makes it easier to translate applications: you just create multiple lists in different languages but with the same item references, then load the list based on the language of the application.

Using this property requires compliance with the following principles:

- To be able to store the reference, the field or variable data source must be of the Number type (regardless of the type of value displayed in the list). The [expression](#) property is automatically set.
- Valid and unique references must be associated with list items.

## JSON Grammar

### Name Data Type Possible Values

saveAsString	"value", "reference"
--------------	----------------------

## Objects Supported

[Input - List Box Column](#)

## Selected Items

Collection or entity selection list boxes

Specifies a variable or expression that will be assigned the elements or entities selected by the user.

- for a collection list box, you must use a collection variable or an assignable expression that accepts collections,
- for an entity selection list box, an entity selection object is built. You must use an object variable or an assignable expression that accepts objects.

This property is "read-only", it is automatically updated according to user actions in the list box. You cannot edit its value to modify the list box selection status.

## JSON Grammar

### Name Data Type Possible Values

selectedItemsSource	string	Collection expression
---------------------	--------	--------------------------

## Objects Supported

[List Box](#)

## Selection Name

## Named selection list boxes

Specifies the named selection to be used. You must enter the name of a valid named selection. It can be a process or interprocess named selection. The contents of the list box will be based on this selection. The named selection chosen must exist and be valid at the time the list box is displayed, otherwise the list box will be displayed blank.

Named selections are ordered lists of records. They are used to keep the order and current record of a selection in memory. For more information, refer to **Named Selections** section in the *4D Language Reference manual*.

## JSON Grammar

Name	Data Type	Possible Values
namedSelection	string	Named selection name

## Objects Supported

[List Box](#)

# Display

## Alpha Format

Alpha formats control the way the alphanumeric fields and variables appear when displayed or printed. Here is a list of formats provided for alphanumeric fields:



You can choose a format from this list or use any custom format. The default list contains formats for some of the most common alpha fields that require formats: US telephone numbers (local and long distance), Social Security numbers, and zip codes. You can also enter a custom format name set in the Filters and formats editor of the tool box. In this case, the format cannot be modified in the object properties. Any custom formats or filters that you have created are automatically available, preceded by a vertical bar (|).

The number sign (#) is the placeholder for an alphanumeric display format. You can include the appropriate dashes, hyphens, spaces, and any other punctuation marks that you want to display. You use the actual punctuation marks you want and the number sign for each character you want to display.

For example, consider a part number with a format such as "RB-1762-1".

The alpha format would be:

##-#####-#

When the user enters "RB17621," the field displays:

RB-1762-1

The field actually contains "RB17621".

If the user enters more characters than the format allows, 4D displays the last characters. For example, if the format is:

(#####)

and the user enters "proportion", the field displays:

(portion)

The field actually contains "proportion". 4D accepts and stores the entire entry no matter what the display format. No information is lost.

## JSON Grammar

Name	Data Type	Possible Values
textFormatString		"### #####", "(###) ###-####", "###-##-####", "### ##-##-", "00000", custom formats

## Objects Supported

[Drop-down List](#) - [Combo Box](#) - [List Box Column](#) - [List Box Footer](#)

## Date Format

Date formats control the way dates appear when displayed or printed. For data entry, you enter dates in the MM/DD/YYYY format, regardless of the display format you have chosen.

Unlike [Alpha](#) and [Number](#) formats, display formats for dates must only be selected among the 4D built-in formats.

The table below shows choices available:

Format name	JSON String	Example (US system)
System date short	- (default)	03/25/20
System date abbreviated (1)	systemMedium	Wed, Mar 25, 2020
System date long	systemLong	Wednesday, March 25, 2020
RFC 822	rfc822	Tue, 25 Mar 2020 22:00:00 GMT
Short Century	shortCentury	03/25/20 but 04/25/2032 (2)
Internal date long	long	March 25, 2020
Internal date abbreviated (1)	abbreviated	Mar 25, 2020
Internal date short	short	03/25/2020
ISO Date Time (3)	iso8601	2020-03-25T00:00:00

(1) To avoid ambiguity and in accordance with current practice, the abbreviated date formats display "jun" for June and "jul" for July. This particularity only applies to French versions of 4D.

(2) The year is displayed using two digits when it belongs to the interval (1930;2029) otherwise it will be displayed using four digits. This is by default but it can be modified using the [SET DEFAULT CENTURY](#) command.

(3) The `ISO Date Time` format corresponds to the XML date and time representation standard (ISO8601). It is mainly intended to be used when importing/exporting data in XML format and in Web Services.

Regardless of the display format, if the year is entered with two digits then 4D assumes the century to be the 21st if the year belongs to the interval (00;29) and the 20th if it belongs to the interval (30;99). This is the default setting but it can be modified using the [SET DEFAULT CENTURY](#) command.

## JSON Grammar

Name	Data Type	Possible Values
		"systemShort", "systemMedium", "systemLong", "iso8601", "rfc822", dateFormatString "short", "shortCentury", "abbreviated", "long", "blankIfNull" (can be combined with the other possible values)

## Objects Supported

[Combo Box](#) - [Drop-down List](#) - [Input](#) - [List Box Column](#) - [List Box Footer](#)

# Number Format

Number fields include the Integer, Long integer, Integer 64 bits, Real and Float types.

Number formats control the way numbers appear when displayed or printed. For data entry, you enter only the numbers (including a decimal point or minus sign if necessary), regardless of the display format you have chosen.

4D provides various default number formats.

## Placeholders

In each of the number display formats, the number sign (#), zero (0), caret (^), and asterisk (\*) are used as placeholders. You create your own number formats by using one placeholder for each digit you expect to display.

### Placeholder Effect for leading or trailing zero

#	Displays nothing
0	Displays 0
^	Displays a space (1)
*	Displays an asterisk

(1) The caret (^) generates a space character that occupies the same width as a digit in most fonts.

For example, if you want to display three-digit numbers, you could use the format ###. If the user enters more digits than the format allows, 4D displays <<< in the field to indicate that more digits were entered than the number of digits specified in the display format.

If the user enters a negative number, the leftmost character is displayed as a minus sign (unless a negative display format has been specified). If ##0 is the format, minus 26 is displayed as -26 and minus 260 is displayed as <<< because the minus sign occupies a placeholder and there are only three placeholders.

No matter what the display format, 4D accepts and stores the number entered in the field. No information is lost.

Each placeholder character has a different effect on the display of leading or trailing zeros. A leading zero is a zero that starts a number before the decimal point; a trailing zero is a zero that ends a number after the decimal point.

Suppose you use the format ##0 to display three digits. If the user enters nothing in the field, the field displays 0. If the user enters 26, the field displays 26.

## Separator characters

The numeric display formats (except for scientific notations) are automatically based on regional system parameters. 4D replaces the “.” and “,” characters by, respectively, the decimal separator and the thousand separator defined in the operating system. The period and comma are thus considered as placeholder characters, following the example of 0 or #.

On Windows, when using the decimal separator key of the numeric keypad, 4D makes a distinction depending on the type of field where the cursor is located:

- in a Real type field, using this key will insert the decimal separator

- defined in the system,
- in any other type of field, this key inserts the character associated with the key, usually a period (.) or comma (,).

## Decimal points and other display characters

You can use a decimal point in a number display format. If you want the decimal to display regardless of whether the user types it in, it must be placed between zeros.

You can use any other characters in the format. When used alone, or placed before or after placeholders, the characters always appear. For example, if you use the following format:

`$##0`

a dollar sign always appears because it is placed before the placeholders.

If characters are placed between placeholders, they appear only if digits are displayed on both sides. For example, if you define the format:

`###.##0`

the point appears only if the user enters at least four digits.

Spaces are treated as characters in number display formats.

## Formats for positive, negative, and zero

A number display format can have up to three parts allowing you to specify display formats for positive, negative, and zero values. You specify the three parts by separating them with semicolons as shown below:

Positive;Negative;Zero

You do not have to specify all three parts of the format. If you use just one part, 4D uses it for all numbers, placing a minus sign in front of negative numbers.

If you use two parts, 4D uses the first part for positive numbers and zero and the second part for negative numbers. If you use three parts, the first is for positive numbers, the second for negative numbers, and the third for zero.

The third part (zero) is not interpreted and does not accept replacement characters. If you enter `###;###;#`, the zero value will be displayed "#". In other words, what you actually enter is what will be displayed for the zero value.

Here is an example of a number display format that shows dollar signs and commas, places negative values in parentheses, and does not display zeros:

`$###,##0.00;($###,##0.00);`

Notice that the presence of the second semicolon instructs 4D to use nothing to display zero. The following format is similar except that the absence of the second semicolon instructs 4D to use the positive number format for zero:

`$###,##0.00;($###,##0.00)`

In this case, the display for zero would be \$0.00.

## Scientific notation

If you want to display numbers in scientific notation, use the **ampersand** (&) followed by a number to specify the number of digits you want to display. For example, the format:

&3

would display 759.62 as:

7.60e+2

The scientific notation format is the only format that will automatically round the displayed number. Note in the example above that the number is rounded up to 7.60e+2 instead of truncating to 7.59e+2.

## Hexadecimal formats

You can display a number in hexadecimal using the following display formats:

- `&x`: This format displays hexadecimal numbers using the "0xFFFF" format.
- `&$`: This format displays hexadecimal numbers using the "\$FFF" format.

## XML notation

The `&xml` format will make a number compliant with XML standard rules. In particular, the decimal separator character will be a period "." in all cases, regardless of the system settings.

## Displaying a number as a time

You can display a number as a time (with a time format) by using `&/` followed by a digit. Time is determined by calculating the number of seconds since midnight that the value represents. The digit in the format corresponds to the order in which the time format appears in the Format drop-down menu.

For example, the format:

&/5

corresponds to the 5th time format in the pop-up menu, specifically the AM/PM time. A number field with this format would display 25000 as:

6:56 AM

## Examples

The following table shows how different formats affect the display of numbers. The three columns — Positive, Negative, and Zero — each show how 1,234.50, -1,234.50, and 0 would be displayed.

Format Entered	Positive	Negative	Zero
###	<<<	<<<	
###+#	1234	<<<<	
###+###+#	1234	-1234	
###+##+.##	1234.5	-1234.5	
###+#0.00	1234.50	-1234.50	0.00
###+##0	1234	-1234	0
+###+##0;-###+##0;0	+1234	-1234	0
###+##0DB;###+##0CR;0	1234DB	1234CR	0
###+##0;(####+#0)	1234	(1234)	0
###,##0	1,234	-1,234	0
##,##0.00	1,234.50	-1,234.50	0.00
¥^¥^¥^¥^¥^¥^¥^¥^	1234	-1234	
¥^¥^¥^¥^¥^¥^¥^0	1234	-1234	0
¥^¥^¥^,¥^¥^0	1,234	-1,234	0
¥^¥^,¥^¥^0.00	1,234.50	-1,234.50	0.00
*****	***1234	**-1234	*****
*****0	***1234	**-1234	*****0
**,**0	**1,234	*-1,234	*****0
**,**0.00	*1,234.50	-1,234.50	*****0.00
\$*,**0.00;-\$*,**0.00	\$1,234.50	-\$1,234.50	*****0.00
\$¥^¥^¥^¥^¥^0	\$ 1234	-\$1234	\$ 0
\$¥^¥^¥^¥^0;-\$¥^¥^¥^0	\$1234	-\$1234	\$ 0
\$¥^¥^¥^¥^0 ;(\$¥^¥^¥^0)	\$1234	(\$1234)	\$ 0
\$¥^,¥^¥^0.00 ;(\$¥^,¥^¥^0.00)	\$1,234.50	(\$1,234.50)	\$ 0.00
&2	1.2e+3	-1.2e+3	0.0e+0
&5	1.23450e+3	-1.23450e+3	0.00000
&xml	1234.5	-1234.5	0

## JSON Grammar

Name	Data Type	Possible Values
numberFormatString		Numbers (including a decimal point or minus sign if necessary)

## Objects Supported

[Combo Box](#) - [Drop-down List](#) - [Input](#) - [List Box Column](#) - [List Box Footer](#) - [Progress Indicators](#)

## Picture Format

Picture formats control how pictures appear when displayed or printed. For data entry, the user always enters pictures by pasting them from the Clipboard or by drag and drop, regardless of the display format.

The truncation and scaling options do not affect the picture itself. The contents of a Picture field are always saved. Only the display on the particular form is affected by the picture display format.

## Scaled to fit

JSON grammar: "scaled"

The **Scaled to fit** format causes 4D to resize the picture to fit the dimensions of the area.



### Truncated (centered and non-centered)

JSON grammar: "truncatedCenter" / "truncatedTopLeft"

The **Truncated (centered)** format causes 4D to center the picture in the area and crop any portion that does not fit within the area. 4D crops equally from each edge and from the top and bottom.

The **Truncated (non-centered)** format causes 4D to place the upper-left corner of the picture in the upper-left corner of the area and crop any portion that does not fit within the area. 4D crops from the right and bottom.

When the picture format is **Truncated (non-centered)**, it is possible to add scroll bars to the input area.

### Scaled to fit (proportional) and Scaled to fit centered (proportional)

JSON grammar: "proportionalTopLeft" / "proportionalCenter"

When you use **Scaled to fit (proportional)**, the picture is reduced proportionally on all sides to fit the area created for the picture. The **Scaled to fit centered (proportional)** option does the same, but centers the picture in the picture area.

If the picture is smaller than the area set in the form, it will not be modified. If the picture is bigger than the area set in the form, it is proportionally reduced. Since it is proportionally reduced, the picture will not appear distorted.

If you have applied the **Scaled to fit centered (proportional)** format, the picture is also centered in the area:

### Replicated

JSON grammar: "tiled"

When the area that contains a picture with the **Replicated** format is enlarged, the picture is not deformed but is replicated as many times as necessary in order to fill the area entirely.

If the field is reduced to a size smaller than that of the original picture, the picture is truncated (non-centered).

## JSON Grammar

Name	Data Type	Possible Values
pictureFormat	string	"truncatedTopLeft", "scaled", "truncatedCenter", "tiled", "proportionalTopLeft", "proportionalCenter"

## Objects Supported

[Input](#) - [List Box Column](#) - [List Box Footer](#)

## Time Format

Time formats control the way times appear when displayed or printed. For data entry, you enter times in the 24-hour HH:MM:SS format or the 12-hour HH:MM:SS AM/PM format, regardless of the display format you have chosen.

Unlike [Alpha](#) and [Number](#) formats, display formats for times must only be selected among the 4D built-in formats.

The table below shows the Time field display formats and gives examples:

Format name	JSON string	Comments	Example for 04:30:25
HH:MM:SS	hh_mm_ss		04:30:25
HH:MM	hh_mm		04:30
Hour Min Sec	HH_MM_SS		4 hours 30 minutes 25 seconds
Hour Min	HH_MM		4 hours 30 minutes
HH:MM AM/PM	hh_mm_am		4:30 a.m.
MM SS	mm_ss	Time expressed as a duration from 00:00:00	270:25
Min Sec	MM_SS	Time expressed as a duration from 00:00:00	270 Minutes 25 Seconds
ISO Date Time	iso8601	Corresponds to the XML standard for representing time-related data. It is mainly intended to be used when importing/exporting data in XML format	0000-00-00T04:30:25
System time short	- (default)	Standard time format defined in the system	04:30:25
System time long abbreviated	systemMedium	macOS only: Abbreviated time format defined in the system.	
	systemLong	Windows: this format is the same as the System time short format	4•30•25 AM
System time long	systemLong	macOS only: Long time format defined in the system. Windows: this format is the same as the System time short format	4:30:25 AM HNEC

## JSON Grammar

Name	Data Type	Possible Values
timeFormatstring		"systemShort", "systemMedium", "systemLong", "iso8601", "hh_mm_ss", "hh_mm", "hh_mm_am", "mm_ss", "HH_MM_SS", "HH_MM", "MM_SS", "blankIfNull" (can be combined with the other possible values)

## Objects Supported

[Combo Box](#) - [Drop-down List](#) - [Input](#) - [List Box Column](#) - [List Box Footer](#)

## Text when False/Text when True

When a [boolean expression](#) is displayed as:

- a text in an [input object](#)
- a ["popup"](#) in a [list box column](#),

... you can select the text to display for each value:

- **Text when True** - the text to be displayed when the value is "true"
- **Text when False** - the text to be displayed when the value is "false"

## JSON Grammar

Name	Data Type	Possible Values
booleanFormatstring		"¥<textWhenTrue>;¥<textWhenFalse>", e.g. "Assigned;Unassigned"

## Objects Supported

[List Box Column](#) - [Input](#)

## Display Type

Used to associate a display format with the column data. The formats provided depends on the variable type (array type list box) or the data/field type (selection and collection type list boxes).

Boolean and number (numeric or integer) columns can be displayed as check boxes. In this case, the [Title](#) property can be defined.

Boolean columns can also be displayed as pop-up menus. In this case, the [Text when False](#) and [Text when True](#) properties must be defined.

## JSON Grammar

Name	Data Type	Possible Values
controlTypestring		<ul style="list-style-type: none"> <li>• <b>number columns</b>: "automatic" (default) or "checkbox"</li> <li>• <b>boolean columns</b>: "checkbox" (default) or "popup"</li> </ul>

## Objects Supported

[List Box Column](#)

## Not rendered

When this property is enabled, the object is not drawn on the form, however it can still be activated.

In particular, this property allows implementing "invisible" buttons. Non-rendered buttons can be placed on top of graphic objects. They remain invisible and do not highlight when clicked, however their action is triggered when they are clicked.

## JSON Grammar

### Name Data Type Possible Values

display boolean true, false

## Objects Supported

[Button - Drop-down List](#)

## Three-States

Allows a check box object to accept a third state. The variable associated with the check box returns the value 2 when the check box is in the third state.

### Three-states check boxes in list box columns

List box columns with a numeric [data type](#) can be displayed as three-states check boxes. If chosen, the following values are displayed:

- 0 = unchecked box,
- 1 = checked box,
- 2 (or any value >0) = semi-checked box (third state). For data entry, this state returns the value 2.
- -1 = invisible check box,
- -2 = unchecked box, not enterable,
- -3 = checked box, not enterable,
- -4 = semi-checked box, not enterable

In this case as well, the [Title](#) property is also available so that the title of the check box can be entered.

## JSON Grammar

### Name Data Type Possible Values

threeState boolean true, false

## Objects Supported

[Check box - List Box Column](#)

## Title

This property is available for a list box column if:

- the [column type](#) is **boolean** and its [display type](#) is "Check Box"
- the [column type](#) is **number** (numeric or integer) and its [display type](#) is "Three-states Checkbox".

In that cases, the title of the check box can be entered using this property.

## JSON Grammar

Name	Data Type	Possible Values
controlTitle	string	Any custom label for the check box

## Objects Supported

[List Box Column](#)

## Truncate with ellipsis

Controls the display of values when list box columns are too narrow to show their full contents.

This option is available for columns with any type of contents, except pictures and objects.

- When the property is enabled (default), if the contents of a list box cell exceed the width of the column, they are truncated and an ellipsis is displayed:

The position of the ellipsis depends on the OS. In the above example (Windows), it is added on the right side of the text. On macOS, the ellipsis is added in the middle of the text.

- When the property is disabled, if the contents of a cell exceed the width of the column, they are simply clipped with no ellipsis added:

The Truncate with ellipsis option is enabled by default and can be specified with list boxes of the Array, Selection, or Collection type.

When applied to Text type columns, the Truncate with ellipsis option is available only if the [Wordwrap](#) option is not selected. When the Wordwrap property is selected, extra contents in cells are handled through the word-wrapping features so the Truncate with ellipsis property is not available.

The Truncate with ellipsis property can be applied to Boolean type columns; however, the result differs depending on the [cell format](#):

- For Pop-up type Boolean formats, labels are truncated with an ellipsis,
- For Check box type Boolean formats, labels are always clipped.

## JSON Grammar

Name	Data Type	Possible Values
truncateMode	string	"withEllipsis", "none"

## Objects Supported

[List Box Column - List Box Header](#)

## Visibility

This property allows hiding the object in the Application environment.

You can handle the Visibility property for most form objects. This property is mainly used to simplify dynamic interface development. In this context, it is often necessary to hide objects programatically during the `On load` event of the form then to display certain objects afterwards. The Visibility property allows inverting this logic by making certain objects invisible by default. The developer can then program their display using the [OBJECT SET VISIBLE](#) command when needed.

## Automatic visibility in list forms

In the context of ["List" forms](#), the Visibility property supports two specific values:

- **If record selected** (JSON name: "selectedRows")
- **If record not selected** (JSON name: "unselectedRows")

This property is only used when drawing objects located in the body of a list form. It tells 4D whether or not to draw the object depending on whether the record being processed is selected/not selected. It allows you to represent a selection of records using visual attributes other than highlight colors:

4D does not take this property into account if the object was hidden using the [OBJECT SET VISIBLE](#) command; in this case, the object remains invisible regardless of whether or not the record is selected.

## JSON Grammar

Name	Data Type	Possible Values
visibility	string	"visible", "hidden", "selectedRows" (list form only), "unselectedRows" (list form only)

## Objects Supported

[4D View Pro area](#) - [4D Write Pro area](#) - [Button](#) - [Button Grid](#) - [Check Box](#) - [Combo Box](#) - [Drop-down List](#) - [Group Box](#) - [Hierarchical List](#) - [List Box](#) - [List Box Column](#) - [List Box Footer](#) - [List Box Header](#) - [Picture Button](#) - [Picture Pop-up Menu](#) - [Plug-in Area](#) - [Progress indicator](#) - [Radio Button](#) - [Spinner](#) - [Splitter](#) - [Static Picture](#) - [Stepper](#) - [Subform](#) - [Tab control](#) - [Text Area](#) - [Web Area](#)

## Wordwrap

For [input](#) objects, available when the [Multiline](#) property is set to "yes".

Manages the display of contents when it exceeds the width of the object.

## Checked for list box/Yes for input

JSON grammar: "normal"

When this option is selected, text automatically wraps to the next line whenever its width exceeds that of the column/area, if the column/area height permits it.

- In single-line columns/areas, only the last word that can be displayed entirely is displayed. 4D inserts line returns; it is possible to scroll the contents of the area by pressing the down arrow key.

- In multiline columns/areas, 4D carries out automatic line returns.

Name	A BLOB is loaded into memory in its entirety. A
Text	<p>A BLOB is loaded into memory in its entirety. A BLOB variable or BLOB array is held and exists in memory only. A BLOB field is loaded into memory from the disk, like the rest of the record to which it belongs.</p> <p>Like the other field types that can retain a large amount of data (such as the Picture field type), BLOB fields are not duplicated in memory when</p>

## Unchecked for list box/No for input

JSON grammar: "none"

When this option is selected, 4D does not do any automatic line returns and the last word that can be displayed may be truncated. In text type areas, carriage returns are supported:

Name	A BLOB is loaded into memory in its entirety. A BLO
Text	<p>A BLOB is loaded into memory in its entirety. A BLO</p> <p>Like the other field types that can retain a large amo</p>

In list boxes, any text that is too long is truncated and displayed with an ellipse (...). In the following example, the Wordwrap option is **checked for the left column** and **unchecked for the right column**:

Note that regardless of the Wordwrap option's value, the row height is not changed. If the text with line breaks cannot be entirely displayed in the column, it is truncated (without an ellipse). In the case of list boxes displaying just a single row, only the first line of text is displayed:

## Automatic for input (default option)

JSON grammar: "automatic"

- In single-line areas, words located at the end of lines are truncated and there are no line returns.
- In multiline areas, 4D carries out automatic line returns.

Name	A BLOB is loaded into memory in its entirety. A BLO
Text	<p>A BLOB is loaded into memory in its entirety. A BLOB variable or BLOB array is held and exists in memory only. A BLOB field is loaded into memory from the disk, like the rest of the record to which it belongs.</p> <p>Like the other field types that can retain a large amount of data (such as the Picture field type), BLOB fields are not duplicated in memory when</p>

## JSON Grammar

Name	Data Type	Possible Values
wordwrap	string	"automatic" (excluding list box), "normal", "none"

## Objects Supported

[Input](#) - [List Box Column](#) - [List Box Footer](#)

# Entry

## Auto Spellcheck

4D includes an integrated and customizable spell-check utility. Text type [inputs](#) can be checked, as well as [4D Write Pro](#) documents.

The Auto Spellcheck property activates the spell-check for each object. When used, a spell-check is automatically performed during data entry. You can also execute the `SPELL CHECKING` 4D language command for each object to be checked.

## JSON Grammar

Name	Data Type	Possible Values
spellcheck	boolean	true, false

## Objects Supported

[4D Write Pro area - Input](#)

## Context Menu

Allows the user access to a standard context menu in the object when the form is executed.

For a picture type [input](#), in addition to standard editing commands (Cut, Copy, Paste and Clear), the menu contains the **Import...** command, which can be used to import a picture stored in a file, as well as the **Save as...** command, which can be used to save the picture to disk. The menu can also be used to modify the display format of the picture: the **Truncated non-centered**, **Scaled to fit** and **Scaled to fit centered** **prop.** options are provided. The modification of the [display format](#) using this menu is temporary; it is not saved with the record.

For a [multi-style](#) text type [input](#), in addition to standard editing commands, the context menu provides the following commands:

- **Fonts...:** displays the font system dialog box
- **Recent fonts:** displays the names of recent fonts selected during the session. The list can store up to 10 fonts (beyond that, the last font used replaces the oldest). By default, this list is empty and the option is not displayed. You can manage this list using the `SET RECENT FONTS` and `FONT LIST` commands.
- commands for supported style modifications: font, size, style, color and background color. When the user modifies a style attribute via this pop-up menu, 4D generates the `On After Edit` form event.

For a [Web Area](#), the contents of the menu depend of the rendering engine of the platform. It is possible to control access to the context menu via the [WA\\_SET PREFERENCE](#) command.

## JSON Grammar

Name	Data Type	Possible Values
contextMenu	string	"automatic" (used if missing), "none"

## Objects Supported

[Input](#) - [Web Area](#) - [4D Write Pro areas](#)

## Enterable

The Enterable attribute indicates whether users can enter values into the object.

Objects are enterable by default. If you want to make a field or an object non-enterable for that form, you can disable the Enterable property for the object. A non-enterable object only displays data. You control the data by methods that use the field or variable name. You can still use the `On Clicked`, `On Double Clicked`, `On Drag Over`, `On Drop`, `On Getting Focus` and `On Losing Focus` form events with non-enterable objects. This makes it easier to manage custom context menus and lets you design interfaces where you can drag-and-drop and select non-enterable variables.

When this property is disabled, any pop-up menus associated with a list box column via a list are disabled.

## JSON Grammar

Name	Data Type	Possible Values
enterable	boolean	true, false

## Objects Supported

[4D Write Pro areas](#) - [Check Box](#) - [Hierarchical List](#) - [Input](#) - [List Box Column](#) - [Progress Bar](#) - [Ruler](#) - [Stepper](#)

## Entry Filter

An entry filter controls exactly what the user can type during data entry. Unlike [required lists](#) for example, entry filters operate on a character-by-character basis. For example, if a part number always consists of two letters followed by three digits, you can use an entry filter to restrict the user to that pattern. You can even control the particular letters and numbers.

An entry filter operates only during data entry. It has no effect on data display after the user deselects the object. In general, you use entry filters and [display formats](#) together. The filter constrains data entry and the format ensures proper display of the value after data entry.

During data entry, an entry filter evaluates each character as it is typed. If the user attempts to type an invalid character (a number instead of a letter, for example), 4D simply does not accept it. The null character remains unchanged until the user types a valid character.

Entry filters can also be used to display required formatting characters so that the user need not enter them. For example, an American telephone number consists of a three-digit area code, followed by a seven-digit number that is broken up into two groups of three and four digits, respectively. A display format can be used to enclose the area code in parentheses and display a dash after the third digit of the telephone number. When such a format is used, the user does not need to enter the parentheses or the dashes.

## Defining an entry filter

Most of the time, you can use one of the [built-in filters](#) of 4D for what you need;

however, you can also create and use custom filters:

- you can directly enter a filter definition string
- or you can enter the name of an entry filter created in the Filters editor in the Toolbox. The names of custom filters you create begin with a vertical bar (|).

For information about creating entry filters, see [Filter and format codes](#).

## Default entry filters

Here is a table that explains each of the entry filter choices in the Entry Filter drop-down list:

Entry Filter	Description
~A	Allow any letters, but convert to uppercase.
&9	Allow only numbers.
&A	Allow only capital letters.
&a	Allow only letters (uppercase and lowercase).
&@	Allow only alphanumeric characters. No special characters.
~a##	State name abbreviation (e.g., CA). Allow any two letters, but convert to uppercase.
!0&9##/#/#/#	Standard date entry format. Display zeros in entry spaces. Allow any numbers.
!0&9 Day: ## Month: ## Year: ##	Custom date entry format. Display zeros in entry spaces. Allow any numbers. Two entries after each word.
!0&9##:##	Time entry format. Limited to hours and minutes. Display zeros in entry spaces. Allow any four numbers, separated by a colon.
!0&9## Hrs ## Mins ## Secs	Time entry format. Display zeros in entry spaces. Allow any two numbers before each word.
!0&9Hrs: ## Mins: ## Secs: ##	Time entry format. Display zeros in entry spaces. Allow any two numbers after each word.
!0&9##-##-##-	Local telephone number format. Display zeros in entry spaces.
##	Allow any number. Three entries, hyphen, four entries.
!_&9(###)!0##- ###	Long distance telephone number. Display underscores in first three entry spaces, zeros in remainder.
!0&9##-##- ##	Long distance telephone number. Display zeros in entry spaces.
!0&9##-##- ##	Allow any number. Three entries, hyphen, three entries, hyphen, four entries.
!0&9##-##- ##	Social Security number. Display zeros in entry spaces. Allow any numbers.
~"A-Z;0-9; ;;;;-"	Uppercase letters and punctuation. Allow only capital letters, numbers, spaces, commas, periods, and hyphens.
&"a-z;0-9; ;;;;-"	Upper and lowercase letters and punctuation. Allow lowercase letters, numbers, spaces, commas, periods, and hyphens.
&"0-9;.;-"	Numbers. Allow only numbers, decimal points, and hyphens (minus sign).

## JSON Grammar

Name	Data Type	Possible Values
entryFilterstring		<ul style="list-style-type: none"><li>• Entry filter code</li><li>or</li><li>• Entry filter name (filter names start with   )</li></ul>

## Objects Supported

[Check Box](#) - [Combo Box](#) - [Hierarchical List](#) - [Input](#) - [List Box Column](#)

## Focusable

When the **Focusable** property is enabled for an object, the object can have the focus (and can thus be activated by the keyboard for instance). It is outlined by a gray dotted line when it is selected — except when the [Hide focus rectangle](#) option has also been selected.

An [input object](#) is always focusable if it has the [Enterable](#) property.

- `Current Member`  
Check box shows focus when selected
- `Current Member`  
Check box is selected but cannot show focus

When the **Focusable** property is selected for a non-enterable object, the user can select, copy or even drag-and-drop the contents of the area.

## JSON Grammar

Name	Data Type	Possible Values
focusable	boolean	true, false

## Objects Supported

[4D Write Pro areas](#) - [Button](#) - [Check Box](#) - [Drop-down List](#) - [Hierarchical List](#) - [Input](#) - [List Box](#) - [Plug-in Area](#) - [Radio Button](#) - [Subform](#)

## Keyboard Layout

This property associates a specific keyboard layout to an [input object](#). For example, in an international application, if a form contains a field whose contents must be entered in Greek characters, you can associate the "Greek" keyboard layout with this field. This way, during data entry, the keyboard configuration is automatically changed when this field has the focus.

By default, the object uses the current keyboard layout.

You can also set and get the keyboard dynamically using the `OBJECT SET KEYBOARD LAYOUT` and `OBJECT Get keyboard layout` commands.

## JSON Grammar

Name	Data Type	Possible Values
keyboardDialect	text	Language code, for example "ar-ma" or "cs". See RFC3066, ISO639 and ISO3166

## Objects Supported

[4D Write Pro areas](#) - [Input](#)

## Multiline

This property is available for [inputs objects](#) containing expressions of the Text type and fields of the Alpha and Text type. It can have three different values: Yes, No, Automatic (default).

### Automatic

- In single-line inputs, words located at the end of lines are truncated and there are no line returns.
- In multiline inputs, 4D carries out automatic line returns:

Name	A BLOB is loaded into memory in its entirety. A BLO
Text	A BLOB is loaded into memory in its entirety. A BLOB variable or BLOB array is held and exists in memory only. A BLOB field is loaded into memory from the disk, like the rest of the record to which it belongs.  Like the other field types that can retain a large amount of data (such as the Picture field type), BLOB fields are not duplicated in memory when

### No

- In single-line inputs, words located at the end of lines are truncated and there are no line returns.
- There are never line returns: the text is always displayed on a single row. If the Alpha or Text field or variable contains carriage returns, the text located after the first carriage return is removed as soon as the area is modified:

Name	A BLOB is loaded into memory in its entirety. A BLO
Text	A BLOB is loaded into memory in its entirety. A BLO

### Yes

When this value is selected, the property is managed by the [Wordwrap](#) option.

## JSON Grammar

Name	Data Type	Possible Values
multilinetext		"yes", "no", "automatic" (default if not defined)

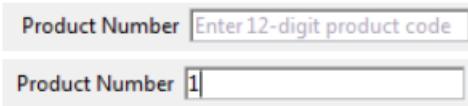
## Objects Supported

### [Input](#)

## Placeholder

4D can display placeholder text in the fields of your forms.

Placeholder text appears as watermark text in a field, supplying a help tip, indication or example for the data to be entered. This text disappears as soon as the user enters a character in the area:



The placeholder text is displayed again if the contents of the field is erased.

A placeholder can be displayed for the following types of data:

- string (text or alpha)
- date and time when the **Blank if null** property is enabled.

You can use an XLIFF reference in the ":xiff:resname" form as a placeholder, for example:

:xiff:PH\_Lastname

You only pass the reference in the "Placeholder" field; it is not possible to combine a reference with static text.

You can also set and get the placeholder text by programming using the [OBJECT SET PLACEHOLDER](#) and [OBJECT Get placeholder](#) commands.

## JSON Grammar

Name	Data Type	Possible Values
placeholderstring		Text to be displayed (grayed out) when the object does not contain any value

## Objects Supported

[Combo Box - Input](#)

## See also

[Help tip](#)

## Selection always visible

This property keeps the selection visible within the object after it has lost the focus. This makes it easier to implement interfaces that allow the text style to be modified (see [Multi-style](#)).

## JSON Grammar

Name	Data Type	Possible Values
showSelection	boolean	true, false

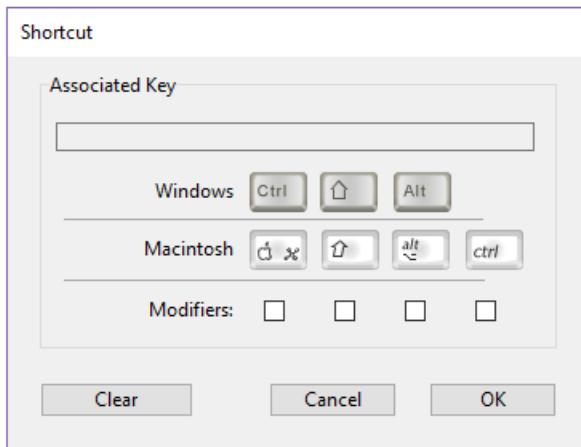
## Objects Supported

[4D Write Pro areas - Input](#)

## Shortcut

This property allows setting special meaning keys (keyboard shortcuts) for [buttons](#), [radio buttons](#), and [checkboxes](#). They allow the user to use the control using the keyboard instead of having to use the mouse.

You can configure this option by clicking the [...] button in the Shortcuts property in the Property List.



You can also assign a shortcut to a custom menu command. If there is a conflict between two shortcuts, the active object has priority. For more information about associating shortcuts with menus, refer to [Setting menu properties](#).

To view a list of all the shortcuts used in the 4D Design environment, see the [Shortcuts Page](#) in the Preferences dialog box.

## JSON Grammar

Name	Data Type	Possible Values
shortcutAccel	boolean	true, false (Ctrl Windows/Command macOS)
shortcutAlt	boolean	true, false
shortcutCommand	boolean	true, false
shortcutControl	boolean	true, false (macOS Control)
shortcutShift	boolean	true, false <ul style="list-style-type: none"><li>• any character key: "a", "b"...</li><li>• [F1]" -&gt; "[F15]", "[Return]", "[Enter]", "[Backspace]",</li></ul>
shortcutKey	string	"[Tab]", "[Esc]", "[Del]", "[Home]", "[End]", "[Help]", "[Page up]", "[Page down]", "[left arrow]", "[right arrow]", "[up arrow]", "[down arrow]"

## Objects Supported

[Button](#) - [Check Box](#) - [Picture Button](#) - [Radio Button](#)

## Single-Click Edit

Enables direct passage to edit mode in list boxes.

When this option is enabled, list box cells switch to edit mode after a single user click, regardless of whether or not this area of the list box was selected beforehand. Note that this option allows cells to be edited even when the list box [selection mode](#) is set to "None".

When this option is not enabled, users must first select the cell row and then click on a cell in order to edit its contents.

## JSON Grammar

Name	Data Type	Possible Values
singleClickEdit	boolean	true, false

## Objects Supported

[List Box](#)

## Footers

### Display Footers

This property is used to display or hide [list box column footers](#). There is one footer per column; each footer is configured separately.

#### JSON Grammar

Name	Data Type	Possible Values
showFooters	boolean	true, false

#### Objects Supported

##### [List Box](#)

## Height

This property is used to set the row height for a list box footer in **pixels** or **text lines** (when displayed). Both types of units can be used in the same list box:

- *Pixel* - the height value is applied directly to the row concerned, regardless of the font size contained in the columns. If a font is too big, the text is truncated. Moreover, pictures are truncated or resized according to their format.
- *Line* - the height is calculated while taking into account the font size of the row concerned.
  - If more than one size is set, 4D uses the biggest one. For example, if a row contains "Verdana 18", "Geneva 12" and "Arial 9", 4D uses "Verdana 18" to determine the row height (for instance, 25 pixels). This height is then multiplied by the number of rows defined.
  - This calculation does not take into account the size of pictures nor any styles applied to the fonts.
  - In macOS, the row height may be incorrect if the user enters characters that are not available in the selected font. When this occurs, a substitute font is used, which may cause variations in size.

This property can also be set dynamically using the [LISTBOX SET FOOTERS HEIGHT](#) command.

Conversion of units: When you switch from one unit to the other, 4D converts them automatically and displays the result in the Property List. For example, if the font used is "Lucida grande 24", a height of "1 line" is converted to "30 pixels" and a height of "60 pixels" is converted to "2 lines".

Note that converting back and forth may lead to an end result that is different from the starting value due to the automatic calculations made by 4D. This is illustrated in the following sequences:

(font Arial 18): 52 pixels -> 2 lines -> 40 pixels (font Arial 12): 3 pixels -> 0.4 line rounded up to 1 line -> 19 pixels

#### JSON Example

```
"List Box": {  
    "type": "listbox",  
    "showFooters": true,  
    "footerHeight": "44px",  
    ...  
}
```

## JSON Grammar

Name	Data Type	Possible Values
footerHeight	string	positive decimal+px   em

## Objects Supported

### [List Box](#)

## See also

[Headers](#) - [List box footers](#)

## Gridlines

### Horizontal Line Color

Defines the color of the horizontal lines in a list box (gray by default).

#### JSON Grammar

Name	Data Type	Possible Values
horizontalLineStrokecolor		any css value, "transparent", "automatic"

#### Objects Supported

[List Box](#)

### Vertical Line Color

Defines the color of the vertical lines in a list box (gray by default).

#### JSON Grammar

Name	Data Type	Possible Values
verticalLineStrokecolor		any css value, "transparent", "automatic"

#### Objects Supported

[List Box](#)

## Headers

### Display Headers

This property is used to display or hide [list box column headers](#). There is one header per column; each header is configured separately.

#### JSON Grammar

Name	Data Type	Possible Values
showHeaders	boolean	true, false

#### Objects Supported

##### [List Box](#)

## Height

This property is used to set the row height for a list box header in **pixels** or **text lines** (when displayed). Both types of units can be used in the same list box:

- *Pixel* - the height value is applied directly to the row concerned, regardless of the font size contained in the columns. If a font is too big, the text is truncated. Moreover, pictures are truncated or resized according to their format.
- *Line* - the height is calculated while taking into account the font size of the row concerned.
  - If more than one size is set, 4D uses the biggest one. For example, if a row contains "Verdana 18", "Geneva 12" and "Arial 9", 4D uses "Verdana 18" to determine the row height (for instance, 25 pixels). This height is then multiplied by the number of rows defined.
  - This calculation does not take into account the size of pictures nor any styles applied to the fonts.
  - In macOS, the row height may be incorrect if the user enters characters that are not available in the selected font. When this occurs, a substitute font is used, which may cause variations in size.

This property can also be set dynamically using the [LISTBOX SET HEADERS HEIGHT](#) command.

Conversion of units: When you switch from one unit to the other, 4D converts them automatically and displays the result in the Property List. For example, if the font used is "Lucida grande 24", a height of "1 line" is converted to "30 pixels" and a height of "60 pixels" is converted to "2 lines".

Note that converting back and forth may lead to an end result that is different from the starting value due to the automatic calculations made by 4D. This is illustrated in the following sequences:

- (font Arial 18)\*: 52 pixels -> 2 lines -> 40 pixels
- (font Arial 12)\*: 3 pixels -> 0.4 line rounded up to 1 line -> 19 pixels

#### JSON Example

```
"List Box": {  
    "type": "listbox",  
    "showHeaders": true,  
    "headerHeight": "22px",  
    ...  
}
```

## JSON Grammar

Name	Data Type	Possible Values
headerHeight	string	positive decimal+px   em

## Objects Supported

[List Box](#)

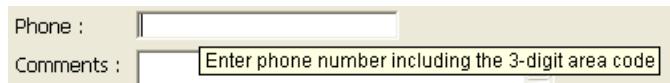
## See also

[Footers](#) - [List box headers](#)

# Help

## Help Tip

This property allows associating help messages with active objects in your forms. They can be displayed at runtime:



- The display delay and maximum duration of help tips can be controlled using the `Tips delay` and `Tips duration` selectors of the [SET DATABASE PARAMETER](#) command.
- Help tips can be globally disabled or enabled for the application using the `Tips enabled` selector of the [SET DATABASE PARAMETER](#) command.

You can either:

- designate an existing help tip, previously specified in the [Help tips](#) editor of 4D.
- or enter the help message directly as a string. This allows you to take advantage of XLIFF architecture. You can enter an XLIFF reference here in order to display a message in the application language (for more information about XLIFF, refer to [Appendix B: XLIFF architecture](#)). You can also use 4D references ([see Using references in static text](#)).

In macOS, displaying help tips is not supported in pop-up type windows.

## JSON Grammar

Name	Data Type	Possible Values
tooltip	text	additional information to help a user

## Objects Supported

[Button](#) - [Button Grid](#) - [Check Box](#) - [Drop-down List](#) - [Combo Box](#) - [Hierarchical List](#) - [List Box Header](#) - [List Box Footer](#) - [Picture Button](#) - [Picture Pop-up menu](#) - [Radio Button](#)

## Other help features

You can also associate help messages with form objects in two other ways:

- at the level of the database structure (fields only). In this case, the help tip of the field is displayed in every form where it appears. For more information, refer to "Help Tips" in [Field properties](#).
- using the [OBJECT SET HELP TIP](#) command, for the current process.

When different tips are associated with the same object in several locations, the following priority order is applied:

1. structure level (lowest priority)
2. form editor level
3. [OBJECT SET HELP TIP](#) command (highest priority)

## See also

[Placeholder](#)

## Hierarchy

### Hierarchical List Box

Array type list boxes

This property specifies that the list box must be displayed in hierarchical form. In the JSON form, this feature is triggered [when the `dataSource` property value is an array](#), i.e. a collection.

Additional options (**Variable 1...10**) are available when the *Hierarchical List Box* option is selected, corresponding to each `dataSource` array to use as break column. Each time a value is entered in a field, a new row is added. Up to 10 variables can be specified. These variables set the hierarchical levels to be displayed in the first column.

See [Hierarchical list boxes](#)

### JSON Grammar

Name	Data Type	Possible Values
<code>datasource</code>	string array	Collection of array names defining the hierarchy

### Objects Supported

[List Box](#)

## List Box

### Columns

Collection of columns of the list box.

#### JSON Grammar

Name	Data Type	Possible Values
columns	collection of column objects	Contains the properties for the list box columns

For a list of properties supported by column objects, please refer to the [Column Specific Properties](#) section.

### Objects Supported

[List Box](#)

## Detail Form Name

Selection type list box

Specifies the form to use for modifying or displaying individual records of the list box.

The specified form is displayed:

- when using `Add Subrecord` and `Edit Subrecord` standard actions applied to the list box (see [Using standard actions](#)),
- when a row is double-clicked and the `Double-click on Row` property is set to "Edit Record" or "Display Record".

#### JSON Grammar

Name	Data Type	Possible Values
detailForm	string	<ul style="list-style-type: none"><li>• Name (string) of table or project form</li><li>• POSIX path (string) to a .json file describing the form</li><li>• Object describing the form</li></ul>

### Objects Supported

[List Box](#)

## Double-click on row

Selection type list box

Sets the action to be performed when a user double-clicks on a row in the list box. The available options are:

- **Do nothing** (default): Double-clicking a row does not trigger any automatic action.
- **Edit Record**: Double-clicking a row displays the corresponding record in the detail form defined [for the list box](#). The record is opened in read-write mode so it can

be modified.

- **Display Record:** Identical to the previous action, except that the record is opened in read-only mode so it cannot be modified.

Double-clicking an empty row is ignored in list boxes.

Regardless of the action selected/chosen, the `On Double clicked` form event is generated.

For the last two actions, the `On Open Detail` form event is also generated. The `On Close Detail` is then generated when a record displayed in the detail form associated with the list box is about to be closed (regardless of whether or not the record was modified).

## JSON Grammar

Name	Data Type	Possible Values
<code>doubleClickInRowAction</code>	string	"editSubrecord", "displaySubrecord"

## Objects Supported

### List Box

## Highlight Set

`Selection type list box`

This property is used to specify the set to be used to manage highlighted records in the list box (when the **Arrays** data source is selected, a Boolean array with the same name as the list box is used).

4D creates a default set named *ListBoxSetN* where *N* starts at 0 and is incremented according to the number of list boxes in the form. If necessary, you can modify the default set. It can be a local, process or interprocess set (we recommend using a local set, for example `$LBSets`, in order to limit network traffic). It is then maintained automatically by 4D. If the user selects one or more rows in the list box, the set is updated immediately. If you want to select one or more rows by programming, you can apply the commands of the "Sets" theme to this set.

- The highlighted status of the list box rows and the highlighted status of the table records are completely independent.
- If the "Highlight Set" property does not contain a name, it will not be possible to make selections in the list box.

## JSON Grammar

Name	Data Type	Possible Values
<code>highlightSet</code>	string	Name of the set

## Objects Supported

### List Box

## Locked columns and static columns

Locked columns and static columns are two separate and independent functionalities in list boxes:

- Locked columns always stay displayed to the left of the list box; they do not scroll horizontally.
- Static columns cannot be moved by drag and drop within the list box.

You can set static and locked columns by programming, refer to [List Box](#) in the [4D Language Reference](#) manual.

These properties interact as follows:

- If you set columns that are only static, they cannot be moved.
- If you set columns that are locked but not static, you can still change their position freely within the locked area. However, a locked column cannot be moved outside of this locked area.
- If you set all of the columns in the locked area as static, you cannot move these columns within the locked area.
- You can set a combination of locked and static columns according to your needs. For example, if you set three locked columns and one static column, the user can swap the two right-most columns within the locked area (since only the first column is static).

## **Number of Locked Columns**

Number of columns that must stay permanently displayed in the left part of the list box, even when the user scrolls through the columns horizontally.

### **JSON Grammar**

<b>Name</b>	<b>Data Type</b>	<b>Possible Values</b>
lockedColumnCount	integer	minimum: 0

### **Objects Supported**

[List Box](#)

## **Number of Static Columns**

Number of columns that cannot be moved during execution.

### **JSON Grammar**

<b>Name</b>	<b>Data Type</b>	<b>Possible Values</b>
staticColumnCount	integer	minimum: 0

### **Objects Supported**

[List Box](#)

## **Number of Columns**

Sets the number of columns of the list box.

You can add or remove columns dynamically by programming, using commands such as [LISTBOX INSERT COLUMN](#) or [LISTBOX DELETE COLUMN](#).

## JSON Grammar

Name	Data Type	Possible Values
columnCount	integer	minimum: 1

## Objects Supported

### List Box

## Row Control Array

Array type list box

A 4D array controlling the display of list box rows.

You can set the "hidden", "disabled" and "selectable" interface properties for each row in an array-based list box using this array. It can also be designated using the `LISTBOX SET ARRAY` command.

The row control array must be of the Longint type and include the same number of rows as the list box. Each element of the *Row Control Array* defines the interface status of its corresponding row in the list box. Three interface properties are available using constants in the "List Box" constant theme:

Constant Value	Comment
lk row is disabled 2	The corresponding row is disabled. The text and controls such as check boxes are dimmed or grayed out. Enterable text input areas are no longer enterable. Default value: Enabled
lk row is hidden 1	The corresponding row is hidden. Hiding rows only affects the display of the list box. The hidden rows are still present in the arrays and can be managed by programming. The language commands, more particularly <code>LISTBOX Get number of rows</code> or <code>LISTBOX GET CELL POSITION</code> , do not take the displayed/hidden status of rows into account. For example, in a list box with 10 rows where the first 9 rows are hidden, <code>LISTBOX Get number of rows</code> returns 10. From the user's point of view, the presence of hidden rows in a list box is not visibly discernible. Only visible rows can be selected (for example using the Select All command). Default value: Visible
lk row is not selectable 4	The corresponding row is not selectable (highlighting is not possible). Enterable text input areas are no longer enterable unless the <a href="#">Single-Click Edit</a> option is enabled. Controls such as check boxes and lists are still functional however. This setting is ignored if the list box selection mode is "None". Default value: Selectable

To change the status for a row, you just need to set the appropriate constant(s) to the corresponding array element. For example, if you do not want row #10 to be selectable, you can write:

```
aLControlArr{10}:=lk row is not selectable
```

You can define several interface properties at once:

```
aLControlArr{8}:=lk row is not selectable + lk row is disabled
```

Note that setting properties for an element overrides any other values for this element (if not reset). For example:

```
aLControlArr{6}:=lk row is disabled + lk row is not selectable  
//sets row 6 as disabled AND not selectable  
aLControlArr{6}:=lk row is disabled  
//sets row 6 as disabled but selectable again
```

## JSON Grammar

Name	Data Type	Possible Values
rowControlSource	string	Row control array name

## Objects Supported

### List Box

## Selection Mode

Designates the option for allowing users to select rows:

- **None:** Rows cannot be selected if this mode is chosen. Clicking on the list will have no effect unless the [Single-Click Edit](#) option is enabled. The navigation keys only cause the list to scroll; the `On Selection Change` form event is not generated.
- **Single:** One row at a time can be selected in this mode. Clicking on a row will select it. A **Ctrl+click** (Windows) or **Command+click** (macOS) on a row toggles its state (between selected or not).  
The Up and Down arrow keys select the previous/next row in the list. The other navigation keys scroll the list. The `On Selection Change` form event is generated every time the current row is changed.
- **Multiple:** Several rows can be selected simultaneously in this mode.

## JSON Grammar

Name	Data Type	Possible Values
selectionMode	string	"multiple", "single", "none"

## Objects Supported

### List Box

# Objects

## Type

MANDATORY SETTING

This property designates the type of the [active or inactive form object](#).

### JSON Grammar

Name	Data Type	Possible Values
type	string	"button", "buttonGrid", "checkbox", "combo", "dropdown", "groupBox", "input", "line", "list", "listbox", "oval", "picture", "pictureButton", "picturePopup", "plugin", "progress", "radio", "rectangle", "ruler", "spinner", "splitter", "stepper", "subform", "tab", "text", "view", "webArea", "write"

## Objects Supported

[4D View Pro area](#) - [4D Write Pro area](#) - [Button](#) - [Button Grid](#) - [Check Box](#) - [Combo Box](#) - [Drop-down List](#) - [Group Box](#) - [Hierarchical List](#) - [List Box](#) - [List Box Column](#) - [List Box Footer](#) - [List Box Header](#) - [Picture Button](#) - [Picture Pop-up Menu](#) - [Plug-in Area](#) - [Progress indicator](#) - [Radio Button](#) - [Spinner](#) - [Splitter](#) - [Static Picture](#) - [Stepper](#) - [Subform](#) - [Tab control](#) - [Text Area](#) - [Web Area](#)

## Object Name

Each active form object is associated with an object name. Each object name must be unique.

Object names are limited to a size of 255 bytes.

When using 4D's language, you can refer to an active form object by its object name (for more information about this, refer to [Object Properties](#) in the 4D Language Reference manual).

For more information about naming rules for form objects, refer to [Identifiers](#) section.

### JSON Grammar

Name	Data Type	Possible Values
name	string	Any allowed name which does not belong to an already existing object

## Objects Supported

[4D View Pro area](#) - [4D Write Pro area](#) - [Button](#) - [Button Grid](#) - [Check Box](#) - [Combo Box](#) - [Drop-down List](#) - [Group Box](#) - [Hierarchical List](#) - [List Box](#) - [List Box Column](#) - [List Box Footer](#) - [List Box Header](#) - [Picture Button](#) - [Picture Pop-up Menu](#) - [Plug-in Area](#) - [Progress indicator](#) - [Spinner](#) - [Splitter](#) - [Static Picture](#) - [Stepper](#) - [Radio Button](#) - [Subform](#) - [Tab control](#) - [Text Area](#) - [Web Area](#)

## Save value

This property is available when the [Save Geometry](#) option is checked for the form.

This feature is only supported for objects that contribute to the overall geometry of the form. For example, this option is available for check boxes because their value can be used to hide or display additional areas in the window.

Here is the list of objects whose value can be saved:

Object	Saved value
<a href="#">Check Box</a>	Value of associated variable (0, 1, 2)
<a href="#">Drop-down List</a>	Number of selected row
<a href="#">Radio Button</a>	Value of associated variable (1, 0, True or False for buttons according to their type)
<a href="#">Tab control</a>	Number of selected tab

## JSON Grammar

Name	Data Type	Possible Values
memorizeValue	boolean	true, false

## Objects Supported

[Check Box](#) - [Drop-down List](#) - [Radio Button](#) - [Tab control](#)

## Variable or Expression

See also [Expression](#) for Selection and collection type list box columns.

This property specifies the source of the data. Each active form object is associated with an object name and a variable name. The variable name can be different from the object's name. In the same form, you can use the same variable several times while each [object name](#) must be unique.

Variable name size is limited to 31 bytes. See [Identifiers](#) section for more information about naming rules.

The form object variables allow you to control and monitor the objects. For example, when a button is clicked, its variable is set to 1; at all other times, it is 0. The expression associated with a progress indicator lets you read and change the current setting.

Variables or expressions can be enterable or non-enterable and can receive data of the Text, Integer, Numeric, Date, Time, Picture, Boolean, or Object type.

## JSON Grammar

Name	Data Type	Possible Values
dataSource	string, or string array	<ul style="list-style-type: none"><li>• 4D variable, field name, or any expression.</li><li>• Empty string for <a href="#">dynamic variables</a>.</li><li>• String array (collection of array names) for a <a href="#">hierarchical listbox</a> column]</li></ul>

## Expressions

You can use an [expression](#) as data source for an object. Any valid 4D expression is allowed: simple expression, object property, formula, 4D function, project method

name or field using the standard `[Table]Field` syntax. The expression is evaluated when the form is executed and reevaluated for each form event. Note that expressions can be [assignable or non-assignable](#).

If the value entered corresponds to both a variable name and a method name, 4D considers that you are indicating the method.

## Dynamic variables

You can leave it up to 4D to create variables associated with your form objects (buttons, enterable variables, check boxes, etc.) dynamically and according to your needs. To do this, simply leave the "Variable or Expression" property (or `dataSource` JSON field) blank.

When a variable is not named, when the form is loaded, 4D creates a new variable for the object, with a calculated name that is unique in the space of the process variables of the interpreter (which means that this mechanism can be used even in compiled mode). This temporary variable will be destroyed when the form is closed. In order for this principle to work in compiled mode, it is imperative that dynamic variables are explicitly typed. There are two ways to do this:

- You can set the type using the [Expression type](#) property.
- You can use a specific initialization code when the form is loaded that uses, for example, the `VARIABLE TO VARIABLE` command:

```
If (Form event code=On Load)
  var $init : Text
  $Ptr_object:=OBJECT Get pointer(Object named;"comments")
  $init:=""
  VARIABLE TO VARIABLE(Current process;$Ptr_object->,$init)
End if
```

In the 4D code, dynamic variables can be accessed using a pointer obtained with the `OBJECT Get pointer` command. For example:

```
// assign the time 12:00:00 to the variable for the "tstart" object
$p :=OBJECT Get pointer(Object named;"tstart")
$p->:=?12:00:00?
```

There are two advantages with this mechanism:

- On the one hand, it allows the development of "subform" type components that can be used several times in the same host form. Let us take as an example the case of a datepicker subform that is inserted twice in a host form to set a start date and an end date. This subform will use objects for choosing the date of the month and the year. It will be necessary for these objects to work with different variables for the start date and the end date. Letting 4D create their variable with a unique name is a way of resolving this difficulty.
- On the other hand, it can be used to limit memory usage. In fact, form objects only work with process or inter-process variables. However, in compiled mode, an instance of each process variable is created in all the processes, including the server processes. This instance takes up memory, even when the form is not used during the session. Therefore, letting 4D create variables dynamically when loading the forms can save memory.

## Array List Box

For an array list box, the **Variable or Expression** property usually holds the name of the array variable defined for the list box, and for each column. However, you can use a string array (containing arrays names) as `dataSource` value for a list box column to

define a [hierarchical list box](#).

## Objects Supported

[4D View Pro area](#) - [4D Write Pro area](#) - [Button](#) - [Button Grid](#) - [Check Box](#) - [Combo Box](#) - [Drop-down List](#) - [Hierarchical List](#) - [List Box](#) - [List Box Column](#) - [List Box Header](#) - [List Box Footer](#) - [Picture Pop-up Menu](#) - [Plug-in Area](#) - [Progress indicator](#) - [Radio Button](#) - [Spinner](#) - [Splitter](#) - [Stepper](#) - [Subform](#) - [Tab control](#) - [Web Area](#)

## Expression Type

This property is called **Data Type** in the Property List for [selection](#) and [collection](#) type list box columns and for [Drop-down Lists](#) associated to an [object](#) or an [array](#).

Specify the data type for the expression or variable associated to the object. Note that main purpose of this setting is to configure options (such as display formats) available for the data type. It does not actually type the variable itself. In view of project compilation, you must [declare the variable](#).

However, this property has a typing function in the following specific cases:

- **Dynamic variables**: you can use this property to declare the type of dynamic variables.
- **List Box Columns**: this property is used to associate a display format with the column data. The formats provided will depend on the variable type (array type list box) or the data/field type (selection and collection type list boxes). The standard 4D formats that can be used are: Alpha, Numeric, Date, Time, Picture and Boolean. The Text type does not have specific display formats. Any existing custom formats are also available.
- **Picture variables**: you can use this menu to declare the variables before loading the form in interpreted mode. Specific native mechanisms govern the display of picture variables in forms. These mechanisms require greater precision when configuring variables: from now on, they must have already been declared before loading the form — i.e., even before the `On Load` form event — unlike other types of variables. To do this, you need either for the statement `C_PICTURE(varName)` to have been executed before loading the form (typically, in the method calling the `DIALOG` command), or for the variable to have been typed at the form level using the expression type property. Otherwise, the picture variable will not be displayed correctly (only in interpreted mode).

## JSON Grammar

Name	Data Type	Possible Values
dataSourceTypeHint	string	<ul style="list-style-type: none"><li>• <b>standard objects</b>: "integer", "boolean", "number", "picture", "text", "date", "time", "arrayText", "arrayDate", "arrayTime", "arrayNumber", "collection", "object", "undefined"</li><li>• <b>list box columns</b>: "boolean", "number", "picture", "text", "date", "time". <i>Array/selection list box only</i>: "integer", "object"</li></ul>

## Objects Supported

[Check Box](#) - [Combo Box](#) - [Drop-down List](#) - [Input](#) - [List Box Column](#) - [List Box Footer](#) - [Plug-in Area](#) - [Progress indicator](#) - [Radio Button](#) - [Ruler](#) - [Spinner](#) - [Stepper](#) - [Subform](#)

- [Tab Control](#)

## CSS Class

A list of space-separated words used as class selectors in [css files](#).

### JSON Grammar

Name	Data Type	Possible Values
class	string	One string with CSS name(s) separated by space characters

### Objects Supported

[4D View Pro area](#) - [4D Write Pro area](#) - [Button](#) - [Button Grid](#) - [Check Box](#) - [Combo Box](#) - [Drop-down List](#) - [Group Box](#) - [Hierarchical List](#) - [List Box](#) - [Picture Button](#) - [Picture Pop-up Menu](#) - [Plug-in Area](#) - [Radio Button](#) - [Static Picture](#) - [Subform](#) - [Text Area](#) - [Web Area](#)

## Collection or entity selection

To use collection elements or entities to define the row contents of the list box.

Enter an expression that returns either a collection or an entity selection. Usually, you will enter the name of a variable, a collection element or a property that contain a collection or an entity selection.

The collection or the entity selection must be available to the form when it is loaded. Each element of the collection or each entity of the entity selection will be associated to a list box row and will be available as an object through the [This](#) command:

- if you used a collection of objects, you can call **This** in the datasource expression to access each property value, for example `This.\<propertyPath>`.
- if you used an entity selection, you can call **This** in the datasource expression to access each attribute value, for example `This.\<attributePath>`.

If you used a collection of scalar values (and not objects), 4D allows you to display each value by calling **This.value** in the datasource expression.

However in this case you will not be able to modify values or to access the current item object (see below) Note: For information about entity selections, please refer to the [ORDA](#) chapter.

### JSON Grammar

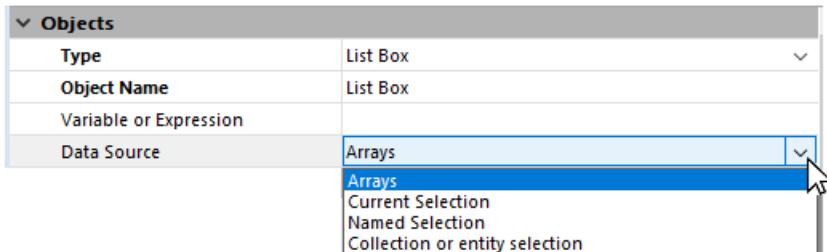
Name	Data Type	Possible Values
dataSource	string	Expression that returns a collection or an entity selection.

### Objects Supported

[List Box](#)

## Data Source

Specify the type of list box.



- **Arrays**(default): use array elements as the rows of the list box.
- **Current Selection**: use expressions, fields or methods whose values will be evaluated for each record of the current selection of a table.
- **Named Selection**: use expressions, fields or methods whose values will be evaluated for each record of a named selection.
- **Collection or Entity Selection**: use collection elements or entities to define the row contents of the list box. Note that with this list box type, you need to define the [Collection or Entity Selection](#) property.

## JSON Grammar

Name	Data Type	Possible Values
listboxType	string	"array", "currentSelection", "namedSelection", "collection"

## Objects Supported

[List Box](#)

## Plug-in Kind

Name of the [plug-in external area](#) associated to the object. Plug-in external area names are published in the manifest.json file of the plug-in.

## JSON Grammar

Name	Data Type	Possible Values
pluginAreaKind	string	Name of the plug-in external area (starts with a % character)

## Objects Supported

[Plug-in Area](#)

## Radio Group

Enables radio buttons to be used in coordinated sets: only one button at a time can be selected in the set.

## JSON Grammar

Name	Data Type	Possible Values
radioGroup	string	Radio group name

## Objects Supported

[Radio Button](#)

## Title

Allows inserting a label on an object. The font and the style of this label can be specified.

You can force a carriage return in the label by using the ¥ character (backslash).



To insert a ¥ in the label, enter "¥¥".

By default, the label is placed in the center of the object. When the object also contains an icon, you can modify the relative location of these two elements using the [Title/Picture Position](#) property.

For application translation purposes, you can enter an XLIFF reference in the title area of a button (see [Appendix B: XLIFF architecture](#)).

## JSON Grammar

### Name Data Type Possible Values

text string any text

## Objects Supported

[Button](#) - [Check Box](#) - [List Box Header](#) - [Radio Button](#) - [Text Area](#)

## Variable Calculation

This property sets the type of calculation to be done in a [column footer](#) area.

The calculation for footers can also be set using the [LISTBOX SET FOOTER CALCULATION](#) 4D command.

There are several types of calculations available. The following table shows which calculations can be used according to the type of data found in each column and indicates the type automatically affected by 4D to the footer variable (if it is not typed by the code):

Calculation	Num	Text	Date	Time	Bool	Pict	footer var type
Minimum	X	X	X	X	X		Same as column type
Maximum	X	X	X	X	X		Same as column type
Sum	X			X	X		Same as column type
Count	X	X	X	X	X	X	Longint
Average	X			X			Real
Standard deviation(*)	X			X			Real
Variance(*)	X			X			Real
Sum squares(*)	X			X			Real
Custom ("none")	X	X	X	X	X	X	Any

(\*) Only for array type list boxes.

Only declared or dynamic [variables](#) can be used to display footer calculations.

Other kinds of [expressions](#) such as `Form.value` are not supported.

Automatic calculations ignore the shown/hidden state of list box rows. If you want to restrict a calculation to only visible rows, you must use a custom calculation.

*Null* values are not taken into account for any calculations.

If the column contains different types of values (collection-based column for example):

- Average and Sum only take numerical elements into account (other element types are ignored).
- Minimum and Maximum return a result according to the usual type list order as defined in the [collection.sort\(\)](#) function.

Using automatic calculations in footers of columns based upon expressions has the following limitations:

- it is **supported** with all list box types when the expression is "simple" (such as `[table]field` or `this.attribute`),
- it is **supported but not recommended** for performance reasons with collection/entity selection list boxes when the expression is "complex" (other than `this.attribute`) and the list box contains a large number of rows,
- it is **not supported** with current selection/named selection list boxes when the expression is "complex". You need to use custom calculations.

When **Custom** ("none" in JSON) is set, no automatic calculations are performed by 4D and you must assign the value of the variable in this area by programming.

## JSON Grammar

Name	Data Type	Possible Values
variableCalculation	string	"none", "minimum", "maximum", "sum", "count", "average", "standardDeviation", "variance", "sumSquare"

## Objects Supported

### [List Box Footer](#)

# Picture

## Pathname

Pathname of a static source picture for a [picture button](#), [picture pop-up Menu](#), or [static picture](#). You must use the POSIX syntax.

The following locations can be used for static pictures:

- in the **Resources** folder of the project. Appropriate when you want to share static pictures between several forms in the project. In this case, the Pathname is "/RESOURCES/<picture path>".
- in an image folder (e.g. named **Images**) within the form folder. Appropriate when the static pictures are used only in the form and/or you want to be able to move or duplicate the whole form within the project or different projects. In this case, the Pathname is "<picture path>" and is resolved from the root of the form folder.
- in a 4D picture variable. The picture must be loaded in memory when the form is executed. In this case, the Pathname is "var:<variableName>".

## JSON Grammar

Name	Data Type	Possible Values
picture	text	Relative or filesystem path in POSIX syntax, or "var:<variableName>" for picture variable

## Objects Supported

[Picture button](#) - [Picture Pop-up Menu](#) - [Static Picture](#)

## Display

### Scaled to fit

JSON grammar: "scaled"

The **Scaled to fit** format causes 4D to resize the picture to fit the dimensions of the area.



### Replicated

JSON grammar: "tiled"

When the area that contains a picture with the **Replicated** format is enlarged, the

picture is not deformed but is replicated as many times as necessary in order to fill the area entirely.

If the field is reduced to a size smaller than that of the original picture, the picture is truncated (non-centered).

## Center / Truncated (non-centered)

JSON grammar: "truncatedCenter" / "truncatedTopLeft"

The **Center** format causes 4D to center the picture in the area and crop any portion that does not fit within the area. 4D crops equally from each edge and from the top and bottom.

The **Truncated (non-centered)** format causes 4D to place the upper-left corner of the picture in the upper-left corner of the area and crop any portion that does not fit within the area. 4D crops from the right and bottom.

When the picture format is **Truncated (non-centered)**, it is possible to add scroll bars to the input area.

## JSON Grammar

Name	Data Type	Possible Values
pictureFormat	string	"scaled", "tiled", "truncatedCenter", "truncatedTopLeft"

## Objects Supported

[Static Picture](#)

## Plug-ins

### Advanced Properties

If advanced options are provided by the author of the plug-in, an **Advanced Properties** button may be enabled in the Property list. In this case, you can click this button to set these options, usually through a custom dialog box.

Because the Advanced properties feature is under the control of the author of the plug-in, information about these Advanced options is the responsibility of the distributor of the plug-in.

### JSON Grammar

Name	Data Type	Possible Values
customPropertiesText		Plugin specific properties, passed to plugin as a JSON string if an object, or as a binary buffer if a base64 encoded string

### Objects Supported

#### [Plug-in Area](#)

# Print

## Print frame

This property handles the print mode for objects whose size can vary from one record to another depending on their contents. These objects can be set to print with either a fixed or variable frame. Fixed frame objects print within the confines of the object as it was created on the form. Variable frame objects expand during printing to include the entire contents of the object. Note that the width of objects printed as a variable size is not affected by this property; only the height varies automatically based on the contents of the object.

You cannot place more than one variable frame object side-by-side on a form. You can place non-variable frame objects on either side of an object that will be printed with a variable size provided that the variable frame object is at least one line longer than the object beside it and that all objects are aligned on the top. If this condition is not respected, the contents of the other fields will be repeated for every horizontal slice of the variable frame object.

The `Print object` and `Print form` commands do not support this property.

The print options are:

- **Variable** option / **Print Variable Frame** checked: 4D enlarges or reduces the form object area in order to print all the subrecords.
- **Fixed (Truncation)** option / **Print Variable Frame** unchecked: 4D only prints the contents that appear in the object area. The form is only printed once and the contents not printed are ignored.
- **Fixed (Multiple Records)** (subforms only): the initial size of the subform area is kept but 4D prints the form several times in order to print all the records.

This property can be set by programming using the `OBJECT SET PRINT VARIABLE FRAME` command.

## JSON Grammar

Name	Data Type	Possible Values
printFrame	string	"fixed", "variable", (subform only) "fixedMultiple"

## Objects Supported

[Input - Subforms](#) (list subforms only) - [4D Write Pro areas](#)

## Range of Values

### Default value

You can assign a default value to be entered in an input object. This property is useful for example when the input [data source](#) is a field: the default value is entered when a new record is first displayed. You can change the value unless the input area has been defined as [non-enterable](#).

The default value can only be used if the [data source type](#) is:

- text/string
- number/integer
- date
- time
- boolean

4D provides stamps for generating default values for the date, time, and sequence number. The date and time are taken from the system date and time. 4D automatically generates any sequence numbers needed. The table below shows the stamp to use to generate default values automatically:

Stamp	Meaning
#D	Current date
#H	Current time
#N	Sequence number

You can use a sequence number to create a unique number for each record in the table for the current data file. A sequence number is a longint that is generated for each new record. The numbers start at one (1) and increase incrementally by one (1). A sequence number is never repeated even if the record it is assigned to is deleted from the table. Each table has its own internal counter of sequence numbers. For more information, refer to the [Autoincrement](#) paragraph.

Do not make confusion between this property and the "[default values](#)" property that allows to fill a list box column with static values.

### JSON Grammar

Name	Data Type	Possible Values
defaultValue	string, number, date, time, boolean	Any value and/or a stamp: "#D", "#H", "#N"

### Objects Supported

#### [Input](#)

### Excluded List

Allows setting a list whose values cannot be entered in the object. If an excluded value is entered, it is not accepted and an error message is displayed.

If a specified list is hierarchical, only the items of the first level are taken into account.

## JSON Grammar

Name	Data Type	Possible Values
excludedList	list	A list of values to be excluded.

### Objects Supported

[Combo Box](#) - [List Box Column](#) - [Input](#)

## Required List

Restricts the valid entries to the items on the list. For example, you may want to use a required list for job titles so that valid entries are limited to titles that have been approved by management.

Making a list required does not automatically display the list when the field is selected. If you want to display the required list, assign the same list to the [Choice List](#) property. However, unlike the [Choice List](#) property, when a required list is defined, keyboard entry is no longer possible, only the selection of a list value using the pop-up menu is allowed. If different lists are defined using the [Choice List](#) and Required List properties, the Required List property has priority.

If a specified list is hierarchical, only the items of the first level are taken into account.

## JSON Grammar

Name	Data Type	Possible Values
requiredList	list	A list of mandatory values.

### Objects Supported

[Combo Box](#) - [List Box Column](#) - [Input](#)

## Resizing Options

### Column Auto-Resizing

When this property is enabled (`rightToLeft` value in JSON), list box columns are automatically resized along with the list box, within the limits of the [minimum](#) and [maximum](#) widths defined.

When this property is disabled (`legacy` value in JSON), only the rightmost column of the list box is resized, even if its width exceeds the maximum value defined.

#### How column auto-resizing works

- As the list box width increases, its columns are enlarged, one by one, starting from right to left, until each reaches its [maximum width](#). Only columns with the [Resizable](#) property selected are resized.
- The same procedure applies when the list box width decreases, but in reverse order (*i.e.*, columns are resized starting from left to right). When each column has reached its [minimum width](#), the horizontal scroll bar becomes active again.
- Columns are resized only when the horizontal scroll bar is not "active"; *i.e.*, all columns are fully visible in the list box at its current size. **Note:** If the horizontal scroll bar is hidden, this does not alter its state: a scroll bar may still be active, even though it is not visible.
- After all columns reach their maximum size, they are no longer enlarged and instead a blank (fake) column is added on the right to fill the extra space. If a fake (blank) column is present, when the list box width decreases, this is the first area to be reduced.

#### About the fake (blank) column

The appearance of the fake column matches that of the existing columns; it will have a fake header and/or footer if these elements are present in the existing list box columns and it will have the same background color(s) applied.

The fake header and/or footer can be clicked but this does not have any effect on the other columns (*e.g.*: no sort is performed); nevertheless, the `On Clicked`, `On Header Click` and `On Footer Click` events are generated accordingly.

If a cell in the fake column is clicked, the [LISTBOX GET CELL POSITION](#) command returns "X+1" for its column number (where X is the number of existing columns).

#### JSON Grammar

Name	Data Type	Possible Values
resizingMode	string	"rightToLeft", "legacy"

#### Objects Supported

[List Box](#)

## Horizontal Sizing

This property specifies if the horizontal size of an object should be moved or resized when a user resizes the form. It can also be set dynamically by the `OBJECT SET RESIZING OPTIONS` language command.

Three options are available:

Option	JSON value	Result
Grow	"grow"	The same percentage is applied to the object's width when the user resizes the width of the window,
Move	"move"	The object is moved the same amount left or right as the width increase when the user resizes the width of the window,
None	"fixed"	The object remains stationary when the form is resized

This property works in conjunction with the [Vertical Sizing](#) property.

### JSON Grammar

Name	Data Type	Possible Values
sizingXstring		"grow", "move", "fixed"

### Objects Supported

[4D View Pro Area](#) - [4D Write Pro Area](#) - [Button](#) - [Button Grid](#) - [Check Box](#) - [Combo Box](#) - [Dropdown list](#) - [Group Box](#) - [Hierarchical List](#) - [Input](#) - [List Box](#) - [Line](#) - [List Box Column](#) - [Oval](#) - [Picture Button](#) - [Picture Pop up menu](#) - [Plug-in Area](#) - [Progress Indicators](#) - [Radio Button](#) - [Ruler](#) - [Rectangle](#) - [Spinner](#) - [Splitter](#) - [Static Picture](#) - [Stepper](#) - [Subform](#) - [Tab control](#) - [Web Area](#)

## Vertical Sizing

This property specifies if the vertical size of an object should be moved or resized when a user resizes the form. It can also be set dynamically by the `OBJECT SET RESIZING OPTIONS` language command.

Three options are available:

Option	JSON value	Result
Grow	"grow"	The same percentage is applied to the object's height when the user resizes the width of the window,
Move	"move"	The object is moved the same amount up or down as the height increase when the user resizes the width of the window,
None	"fixed"	The object remains stationary when the form is resized

This property works in conjunction with the [Horizontal Sizing](#) property.

### JSON Grammar

Name	Data Type	Possible Values
sizingYstring		"grow", "move", "fixed"

### Objects Supported

[4D View Pro Area](#) - [4D Write Pro Area](#) - [Button](#) - [Button Grid](#) - [Check Box](#) - [Combo Box](#) - [Dropdown list](#) - [Group Box](#) - [Hierarchical List](#) - [Input](#) - [List Box](#) - [Line](#) - [List Box Column](#) - [Oval](#) - [Picture Button](#) - [Picture Pop up menu](#) - [Plug-in Area](#) - [Progress Indicators](#) - [Radio Button](#) - [Ruler](#) - [Rectangle](#) - [Spinner](#) - [Splitter](#) - [Static Picture](#) - [Stepper](#) - [Subform](#) - [Tab control](#) - [Web Area](#)

## Pusher

When a splitter object has this property, other objects to its right (vertical splitter) or below it (horizontal splitter) are pushed at the same time as the splitter, with no stop.

Here is the result of a “pusher” splitter being moved:

Field1 :	Field2 :	Field3 :
Alpha	Beta	Omega

Field1 :	Field2 :	Field3 :
Alpha	Beta	Omega

When this property is not applied to the splitter, the result is as follows:

Field1 :	Field2 :	Field3 :
Alpha	Beta	Omega

## JSON Grammar

Name	Data Type	Possible Values
splitterMode	string	"move" (pusher), "resize" (standard)

## Objects Supported

[Splitter](#)

## Resizable

Designates if the size of the column can be modified by the user.

## JSON Grammar

Name	Data Type	Possible Values
resizable	boolean	"true", "false"

## Objects Supported

[List Box Column](#)

## Scale

### Barber shop

Enables the "barber shop" variant for the thermometer.

#### JSON Grammar

Name	Data Type	Possible Values
<code>max</code>	number	NOT passed = enabled; passed = disabled (basic thermometer)

#### Objects Supported

[Barber shop](#)

### Display graduation

Displays/Hides the graduations next to the labels.

#### JSON Grammar

Name	Data Type	Possible Values
showGraduations	boolean	"true", "false"

#### Objects Supported

[Thermometer](#) - [Ruler](#)

### Graduation step

Scale display measurement.

#### JSON Grammar

Name	Data Type	Possible Values
graduationStep	integer	minimum: 0

#### Objects Supported

[Thermometer](#) - [Ruler](#)

### Label Location

Specifies the location of an object's displayed text.

- None - no label is displayed
- Top - Displays labels to the left of or above an indicator
- Bottom - Displays labels to the right of or below an indicator

#### JSON Grammar

Name	Data Type	Possible Values
labelsPlacement	string	"none", "top", "bottom", "left", "right"

## Objects Supported

[Thermometer](#) - [Ruler](#)

## Maximum

Maximum value of an indicator.

- For numeric steppers, this property represent seconds when the object is associated with a time type value and are ignored when it is associated with a date type value.
- To enable [Barber shop thermometers](#), this property must be omitted.

## JSON Grammar

### Name Data Type Possible Values

max number Any number

## Objects Supported

[Thermometer](#) - [Ruler](#) - [Stepper](#)

## Minimum

Minimum value of an indicator. For numeric steppers, this property represent seconds when the object is associated with a time type value and are ignored when it is associated with a date type value.

## JSON Grammar

### Name Data Type Possible Values

min number Any number

## Objects Supported

[Thermometer](#) - [Ruler](#) - [Stepper](#)

## Step

Minimum interval accepted between values during use. For numeric steppers, this property represents seconds when the object is associated with a time type value and days when it is associated with a date type value.

## JSON Grammar

### Name Data Type Possible Values

step integer minimum: 1

## Objects Supported

[Thermometer](#) - [Ruler](#) - [Stepper](#)



## Subform

### Allow Deletion

Specifies if the user can delete subrecords in a list subform.

#### JSON Grammar

Name	Data Type	Possible Values
deletableInList	boolean	true, false (default: true)

#### Objects Supported

[Subform](#)

## Detail Form

You use this property to declare the detail form to use in the subform. It can be:

- a widget, i.e. a page-type subform endowed with specific functions. In this case, the [list subform](#) and [Source](#) properties must be empty or not present.  
You can select a component form name when it is published in the component.  
  
You can generate [components](#) providing additional functionalities through subforms.
- the detail form to associate with the [list subform](#). The detail form can be used to enter or view subrecords. It generally contains more information than the list subform. Naturally, the detail form must belong to the same table as the subform. You normally use an Output form as the list form and an Input form as the detail form. If you do not specify the form to use for full page entry, 4D automatically uses the default Input format of the table.

#### JSON Grammar

Name	Data Type	Possible Values
detailForm	string	Name (string) of table or project form, a POSIX path (string) to a .json file describing the form, or an object describing the form

#### Objects Supported

[Subform](#)

## Double-click on empty row

Action to perform in case of a double-click on an empty line of a list subform. The following options are available:

- Do nothing: Ignores double-click.
- Add Record: Creates a new record in the subform and changes to editing mode. The record will be created directly in the list if the [Enterable in List](#) property is enabled. Otherwise, it will be created in page mode, in the [detail form](#) associated

with the subform.

## JSON Grammar

Name	Data Type	Possible Values
doubleClickInEmptyAreaAction	string	"addSubrecord" or "" to do nothing

## Objects Supported

### Subform

#### See also

[Double click on row](#)

## Double-click on row

List subform

Sets the action to be performed when a user double-clicks on a row in a list subform. The available options are:

- **Do nothing** (default): Double-clicking a row does not trigger any automatic action.
- **Edit Record**: Double-clicking a row displays the corresponding record in the [detail form defined for the list subform](#). The record is opened in read-write mode so it can be modified.
- **Display Record**: Identical to the previous action, except that the record is opened in read-only mode so it cannot be modified.

Regardless of the action selected/chosen, the `On Double clicked` form event is generated.

For the last two actions, the `On Open Detail` form event is also generated. The `On Close Detail` is then generated when a record displayed in the detail form associated with the list box is about to be closed (regardless of whether or not the record was modified).

## JSON Grammar

Name	Data Type	Possible Values
doubleClickInRowAction	string	"editSubrecord", "displaySubrecord"

## Objects Supported

### Subform

#### See also

[Double click on empty row](#)

## Enterable in list

When a list subform has this property enabled, the user can modify record data directly in the list, without having to use the [associated detail form](#).

To do this, simply click twice on the field to be modified in order to switch it

to editing mode (make sure to leave enough time between the two clicks so as not to generate a double-click).

## JSON Grammar

Name	Data Type	Possible Values
enterableInList	boolean	true, false

## Objects Supported

[Subform](#)

## List Form

You use this property to declare the list form to use in the subform. A list subform lets you enter, view, and modify data in other tables.

List subforms can be used for data entry in two ways: the user can enter data directly in the subform, or enter it in an [input form](#). In this configuration, the form used as the subform is referred to as the List form. The input form is referred to as the Detail form.

## JSON Grammar

Name	Data Type	Possible Values
listForm	string	Name (string) of table or project form, a POSIX path (string) to a .json file describing the form, or an object describing the form

## Objects Supported

[Subform](#)

## Source

Specifies the table that the list subform belongs to (if any).

## JSON Grammar

Name	Data Type	Possible Values
table	string	4D table name, or "" if no table.

## Objects Supported

[Subform](#)

## Selection Mode

Designates the option for allowing users to select rows:

- **None:** Rows cannot be selected if this mode is chosen. Clicking on the list will have no effect unless the [Enterable in list](#) option is enabled. The navigation keys only cause the list to scroll; the [On Selection Change](#) form event is not generated.
- **Single:** One row at a time can be selected in this mode. Clicking on a row will

select it. A **Ctrl+click** (Windows) or **Command+click** (macOS) on a row toggles its state (between selected or not).

The Up and Down arrow keys select the previous/next row in the list. The other navigation keys scroll the list. The `On Selection Change` form event is generated every time the current row is changed.

- **Multiple:** Several rows can be selected simultaneously in this mode.

- The selected subrecords are returned by the `GET HIGHLIGHTED RECORDS` command.
- Clicking on the record will select it, but it does not modify the current record.
- A **Ctrl+click** (Windows) or **Command+click** (macOS) on a record toggles its state (between selected or not). The Up and Down arrow keys select the previous/next record in the list. The other navigation keys scroll the list. The `On Selection Change` form event is generated every time the selected record is changed.

## JSON Grammar

Name	Data Type	Possible Values
selectionMode	string	"multiple", "single", "none"

## Objects Supported

[Subform](#)

## Text

### Allow font/color picker

When this property is enabled, the [OPEN FONT PICKER](#) and [OPEN COLOR PICKER](#) commands can be called to display the system font and color picker windows. Using these windows, the users can change the font or color of a form object that has the focus directly by clicking. When this property is disabled (default), the open picker commands have no effect.

#### JSON Grammar

Property	Data Type	Possible Values
allowFontColorPicker	boolean	false (default), true

### Objects Supported

[Input](#)

## Bold

Sets the selected text to appear darker and heavier.

You can set this property using the [OBJECT SET FONT STYLE](#) command.

This is normal text.

**This is bold text.**

#### JSON Grammar

Property	Data Type	Possible Values
fontWeight	text	"normal", "bold"

### Objects Supported

[Button](#) - [Check Box](#) - [Combo Box](#) - [Drop-down List](#) - [Group Box](#) - [Hierarchical List](#) - [Input](#) - [List Box](#) - [List Box Column](#) - [List Box Footer](#) - [List Box Header](#) - [Radio Button](#) - [Text Area](#)

## Italic

Sets the selected text to slant slightly to the right.

You can also set this property via the [OBJECT SET FONT STYLE](#) command.

This is normal text.

*This is text in italics.*

#### JSON Grammar

Name	Data Type	Possible Values
fontStyle	string	"normal", "italic"

## Objects Supported

[Button](#) - [Check Box](#) - [Combo Box](#) - [Drop-down List](#) - [Group Box](#) - [Hierarchical List](#) - [Input](#) - [List Box](#) - [List Box Column](#) - [List Box Footer](#) - [List Box Header](#) - [Radio Button](#) - [Text Area](#)

## Underline

Sets the text to have a line running beneath it.

### JSON Grammar

Name	Data Type	Possible Values
textDecoration	string	"normal", "underline"

## Objects Supported

[Button](#) - [Check Box](#) - [Combo Box](#) - [Drop-down List](#) - [Group Box](#) - [Hierarchical List](#) - [Input](#) - [List Box](#) - [List Box Column](#) - [List Box Footer](#) - [List Box Header](#) - [Radio Button](#) - [Text Area](#)

## Font

This property allows you to specify either the **font theme** or the **font family** used in the object.

**Font theme** and **font family** properties are mutually exclusive. A font theme takes hold of font attributes, including size. A font family allows you to define font name, font size and font color.

### Font Theme

The font theme property designates an automatic style name. Automatic styles determine the font family, font size and font color to be used for the object dynamically according to system parameters. These parameters depend on:

- the platform,
- the system language,
- and the type of form object.

With the font theme, you are guaranteed that titles are always displayed in accordance with the current interface standards of the system. However, their size may vary from one machine to another.

Three font themes are available:

- **normal**: automatic style, applied by default to any new object created in the Form editor.
- **main** and **additional** font themes are only supported by [text areas](#) and [inputs](#). These themes are primarily intended for designing dialog boxes. They refer to font styles used, respectively, for main text and additional information in your interface windows. Here are typical dialog boxes (macOS and Windows) using these font themes:

Font themes manage the font as well as its size and color. You can apply custom style properties (Bold, Italic or Underline) without altering its

functioning.

## JSON Grammar

Name	Data Type	Possible Values
fontTheme	string	"normal", "main", "additional"

## Objects Supported

[Button](#) - [Check Box](#) - [Combo Box](#) - [Drop-down List](#) - [Group Box](#) - [Hierarchical List](#) - [Input](#) - [List Box](#) - [List Box Column](#) - [List Box Footer](#) - [List Box Header](#) - [Radio Button](#) - [Text Area](#)

## Font Family

There are two types of font family names:

- *family-name*: The name of a font-family, like "times", "courier", "arial", etc.
- *generic-family*: The name of a generic-family, like "serif", "sans-serif", "cursive", "fantasy", "monospace".

You can set this using the [OBJECT SET FONT](#) command.

## JSON Grammar

Name	Data Type	Possible Values
fontFamily	string	CSS font family name

4D recommends using only [web safe](#) fonts.

## Objects Supported

[Button](#) - [Check Box](#) - [Combo Box](#) - [Drop-down List](#) - [Group Box](#) - [Hierarchical List](#) - [Input](#) - [List Box](#) - [List Box Column](#) - [List Box Footer](#) - [List Box Header](#) - [Radio Button](#) - [Text Area](#)

## Font Size

Allows defining the object's font size in points.

## JSON Grammar

Name	Data Type	Possible Values
fontSize	integer	Font size in points. Minimum value: 0

## Objects Supported

[Button](#) - [Check Box](#) - [Combo Box](#) - [Drop-down List](#) - [Group Box](#) - [Hierarchical List](#) - [Input](#) - [List Box](#) - [List Box Column](#) - [List Box Footer](#) - [List Box Header](#) - [Radio Button](#) - [Text Area](#)

## Font Color

Designates the font color.

This property also sets the color of object's [border](#) (if any) when "plain" or

"dotted" style is used.

The color can be specified by:

- a color name - like "red"
- a HEX value - like "#ff0000"
- an RGB value - like "rgb(255,0,0)"

You can also set this property using the [OBJECT SET RGB COLORS](#) command.

## JSON Grammar

Name	Data Type	Possible Values
stroke	string	any css value, "transparent", "automatic"

## Objects Supported

[Button](#) - [Check Box](#) - [Combo Box](#) - [Drop-down List](#) - [Group Box](#) - [Hierarchical List](#) - [Input](#) - [List Box](#) - [List Box Column](#) - [List Box Footer](#) - [List Box Header](#) - [Progress Indicators](#) - [Ruler](#) - [Radio Button](#) - [Text Area](#)

## Font Color Expression

Selection and collection/entity selection type list boxes

Used to apply a custom font color to each row of the list box. You must use RGB color values. For more information about this, refer to the description of the [OBJECT SET RGB COLORS](#) command in the 4D Language Reference manual.

You must enter an expression or a variable (array type variables cannot be used). The expression or variable will be evaluated for each row displayed. You can use the constants of the [SET RGB COLORS](#) theme.

You can also set this property using the `LISTBOX SET PROPERTY` command with `1k font color expression` constant.

This property can also be set using a [Meta Info Expression](#).

The following example uses a variable name: enter `CompanyColor` for the **Font Color Expression** and, in the form method, write the following code:

```
CompanyColor:=Choose([Companies]ID;Background color;Light shadow color;  
Foreground color;Dark shadow color)
```

## JSON Grammar

Name	Data Type	Possible Values
rowStrokeSource	string	Font color expression

## Objects Supported

[List Box](#)

## Style Expression

## Selection and collection/entity selection type list boxes

Used to apply a custom character style to each row of the list box or each cell of the column.

You must enter an expression or a variable (array type variables cannot be used). The expression or variable will be evaluated for each row displayed (if applied to the list box) or each cell displayed (if applied to a column). You can use the constants of the [Font Styles](#) theme.

Example:

```
Choose([Companies] ID;Bold;Plain;Italic;Underline)
```

You can also set this property using the `LISTBOX SET PROPERTY` command with `1k font style expression` constant.

This property can also be set using a [Meta Info Expression](#).

## JSON Grammar

Name	Data Type	Possible Values
rowStyleSource	string	Style expression to evaluate for each row/cell.

## Objects Supported

[List Box](#) - [List Box Column](#)

## Horizontal Alignment

Horizontal location of text within the area that contains it.

## JSON Grammar

Name	Data Type	Possible Values
textAlign	string	"right", "center", "left", "automatic", "justify"



- "automatic" is not supported by [check boxes](#) and [radio buttons](#)
- "justify" is only supported by [inputs](#) and [text areas](#)

## Objects Supported

[Button](#) - [Check Box](#) (all styles except Regular and Flat) - [Group Box](#) - [Input](#) - [List Box](#) - [List Box Column](#) - [List Box Header](#) - [List Box Footer](#) - [Radio Button](#) (all styles except Regular and Flat) - [Text Area](#)

## Vertical Alignment

Vertical location of text within the area that contains it.

The **Default** option (`automatic` JSON value) sets the alignment according to the type of data found in each column:

- `bottom` for all data (except pictures) and
- `top` for picture type data.

This property can also be handled by the [OBJECT Get vertical alignment](#) and [OBJECT SET VERTICAL ALIGNMENT](#) commands.

## JSON Grammar

Name	Data Type	Possible Values
verticalAlign	string	"automatic", "top", "middle", "bottom"

## Objects Supported

[List Box](#) - [List Box Column](#) - [List Box Footer](#) - [List Box Header](#)

## Meta Info Expression

Collection or entity selection type list boxes

Specifies an expression or a variable which will be evaluated for each row displayed. It allows defining a whole set of row text attributes. You must pass an **object variable** or an **expression that returns an object**. The following properties are supported:

Property name	Type	Description
stroke	string	Font color. Any CSS color (ex: "#FF00FF"), "automatic", "transparent"
fill	string	Background color. Any CSS color (ex: "#F00FFF"), "automatic", "transparent"
fontStyle	string	"normal", "italic"
fontWeight	string	"normal", "bold"
textDecoration	string	"normal", "underline"
unselectable	boolean	Designates the corresponding row as not being selectable ( <i>i.e.</i> , highlighting is not possible). Enterable areas are no longer enterable if this option is enabled unless the "Single-Click Edit" option is also enabled. Controls such as checkboxes and lists remain functional. This setting is ignored if the list box selection mode is "None". Default value: False.
disabled	boolean	Disables the corresponding row. Enterable areas are no longer enterable if this option is enabled. Text and controls (checkboxes, lists, etc.) appear dimmed or grayed out. Default value: False.

The special "cell" property allows you to apply a set of properties to a single column:

Property name	Type	Description
cell	object	Properties to apply to single column(s)

*columnName* object *columnName* is the object name of the list box column

"stroke", "fill", "fontStyle", "fontWeight", or "textDecoration" property (see above).

**Note:** "unselectable" and "disabled" properties can only be defined at row level. They are ignored if passed in the "cell" object

Style settings made with this property are ignored if other style settings are

already defined through expressions (i.e., [Style Expression](#), [Font Color Expression](#), [Background Color Expression](#)).

## Examples

In a *Color* project method, write the following code:

```
//Color method
//Sets font color for certain rows and background color for Col2 and
Col3 columns
Form.meta:=New object
If(This.ID>5) //ID is an attribute of collection objects/entities
    Form.meta.stroke:="purple"
    Form.meta.cell:=New object("Col2";New object("fill";"black");\
        "Col3";New object("fill";"red"))
Else
    Form.meta.stroke:="orange"
End if
```

**Best Practice:** For optimization reasons, it is usually recommended to create the `meta.cell` object once in the form method:

```
//form method
Case of
    :(Form event code=On Load)
        Form.colStyle:=New object("Col2";New object("fill";"black");\
            "Col3";New object("fill";"red"))
// you can also define other style sets
    Form.colStyle2:=New object("Col2";New object("fill";"green");\
        "Col3";New object("fontWeight";"bold"))
End case
```

Then, the *Color* method would contain:

```
//Color method
...
If(This.ID>5)
    Form.meta.stroke:="purple"
    Form.meta.cell:=Form.colStyle //reuse the same object for better
performance
Else
    Form.meta.stroke:="orange"
    Form.meta.cell:=Form.colStyle2
End if
...
```

## JSON Grammar

Name	Data Type	Possible Values
metaSource	string	Object expression to evaluate for each row/cell.

## Objects Supported

### [List Box](#)

## Multi-style

This property enables the possibility of using specific styles in the selected area. When this option is checked, 4D interprets any `<SPAN>` `HTML` tags found in the area.

By default, this option is not enabled.

## JSON Grammar

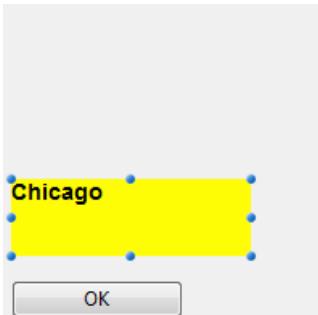
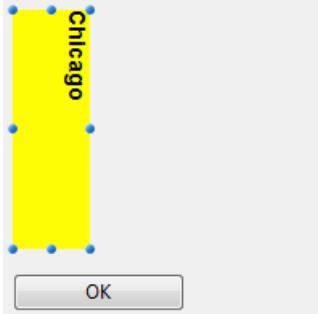
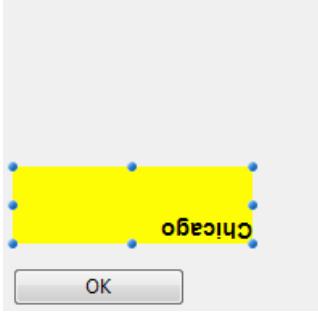
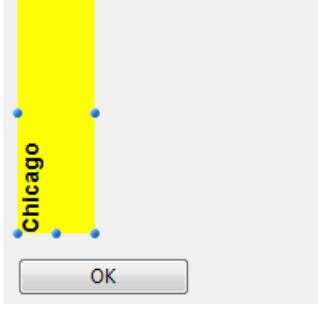
Name	Data Type	Possible Values
styledText	boolean	true, false

## Objects Supported

### [Input - List Box Column](#)

## Orientation

Modifies the orientation (rotation) of a text area. Text areas can be rotated by increments of 90°. Each orientation value is applied while keeping the same lower left starting point for the object:

Orientation value	Result
0 (default)	
90	
180	
270	

In addition to [static text areas](#), [input](#) text objects can be rotated when they are non-[enterable](#). When a rotation property is applied to an input object, the enterable property is removed (if any). This object is then excluded from the entry order.

## JSON Grammar

Name	Data Type	Possible Values
textAngle	number	0, 90, 180, 270

## Objects Supported

[Input](#) (non-enterable) - [Text Area](#)

## Row Font Color Array

Array type list boxes

Allows setting a custom font color to each row of the list box or cell of the column.

The name of a Longint array must be used. Each element of this array corresponds to a row of the list box (if applied to the list box) or to a cell of the column (if applied to a column), so the array must be the same size as the array associated with the column. You can use the constants of the [SET RGB COLORS](#) theme. If you want the cell to inherit the background color defined at the higher level, pass the value -255 to the corresponding array element.

## JSON Grammar

Name	Data Type	Possible Values
rowStrokeSource	string	The name of a longint array

## Objects Supported

[List Box](#) - [List Box Column](#)

## Row Style Array

Array type list boxes

Allows setting a custom font style to each row of the list box or each cell of the column.

The name of a Longint array must be used. Each element of this array corresponds to a row of the list box (if applied to the list box) or to a cell of the column (if applied to a column), so the array must be the same size as the array associated with the column. To fill the array (using a method), use the constants of the [Font Styles](#) theme. You can add constants together to combine styles. If you want the cell to inherit the style defined at the higher level, pass the value -255 to the corresponding array element.

## JSON Grammar

Name	Data Type	Possible Values
rowStyleSource	string	The name of a longint array.

## Objects Supported

[List Box](#) - [List Box Column](#)

## Store with default style tags

This property is only available for a [Multi-style](#) input area. When this property is enabled, the area will store the style tags with the text, even if no modification has been made. In this case, the tags correspond to the default style. When this property is disabled, only modified style tags are stored.

For example, here is a text that includes a style modification:

```
What a beautiful day
```

When the property is disabled, the area only stores the modification. The stored contents are therefore:

```
What a <SPAN STYLE="font-size:13.5pt">beautiful</SPAN> day!
```

When the property is enabled, the area stores all the formatting information. The first generic tag describes the default style then each variation is the subject of a pair of nested tags. The contents stored in the area are therefore:

```
<SPAN STYLE="font-family:'Arial';font-size:9pt;text-align:left;font-weight:normal;font-style:normal;text-decoration:none;color:#000000;background-color:#FFFFFF">What a <SPAN STYLE="font-size:13.5pt">beautiful</SPAN> day!</SPAN>
```

## JSON Grammar

Name	Data Type	Possible Values
storeDefaultStyle	boolean	true, false (default).

## Objects Supported

### [Input](#)

## Text and Picture

### Background pathname

Sets the path of the picture that will be drawn in the background of the object. If the object uses an [icon](#) with [different states](#), the background picture will automatically support the same number of states.

The pathname to enter is similar as for the [Pathname property for static pictures](#).

#### JSON Grammar

Name	Data Type	Possible Values
		Relative path in POSIX syntax. Must be used in customBackgroundPicturestring conjunction with the style property with the "custom" option.

#### Objects Supported

[Custom Button](#) - [Custom Check Box](#) - [Custom Radio Button](#)

### Button Style

General appearance of the button. The button style also plays a part in the availability of certain options.

#### JSON Grammar

Name	Data Type	Possible Values
		"regular", "flat", "toolbar", "bevel", "roundedBevel", "gradientBevel", style text "texturedBevel", "office", "help", "circular", "disclosure", "roundedDisclosure", "custom"

#### Objects Supported

[Button](#) - [Radio Button](#) - [Check Box](#) - [Radio Button](#)

### Horizontal Margin

This property allows setting the size (in pixels) of the horizontal margins of the button. This margin delimits the area that the button icon and title must not surpass.

This parameter is useful, for example, when the background picture contains borders:

With / Without	Example
Without margin	
With 13-pixel margin	

This property works in conjunction with the [Vertical Margin](#) property.

## JSON Grammar

Name	Data Type	Possible Values
customBorderXnumber	number	For use with "custom" style. Minimum: 0

## Objects Supported

[Custom Button](#) - [Custom Check Box](#) - [Custom Radio Button](#)

## Icon Location

Designates the placement of an icon in relation to the form object.

## JSON Grammar

Name	Data Type	Possible Values
iconPlacement	string	"none", "left", "right"

## Objects Supported

[List Box Header](#)

## Icon Offset

Sets a custom offset value in pixels, which will be used when the button is clicked

The title of the button will be shifted to the right and toward the bottom for the number of pixels entered. This allows applying a customized 3D effect when the button is clicked.

## JSON Grammar

Name	Data Type	Possible Values
customOffset	number	minimum: 0

## Objects Supported

[Custom Button](#) - [Custom Check Box](#) - [Custom Radio Button](#)

## Number of States

This property sets the exact number of states present in the picture used as the icon for a [button with icon](#), a [check box](#) or a custom [radio button](#).

The picture can contain from 2 to 6 states.

- 2 states: false, true
- 3 states: false, true, rollover,
- 4 states: false, true, rollover, disabled,
- 5 states (check box and radio button only): false, true, false rollover, true rollover, disabled
- 6 states (check box and radio button only): false, true, false rollover, true rollover, false disabled, true disable.

 NOTE

- "false" means button not clicked/not selected or check box unchecked (variable value=0)
- "true" means button clicked/selected or check box checked (variable value=1)

Each state is represented by a different picture. In the source picture, the states must be stacked vertically:

## JSON Grammar

Name	Data Type	Possible Values
iconFramesNumber	Number of states in the icon picture. Minimum: 1	

## Objects Supported

[Button](#) (all styles except [Help](#)) - [Check Box](#) - [Radio Button](#)

## Picture pathname

Sets the path of the picture that will be used as icon for the object.

The pathname to enter is similar as for the [Pathname property for static pictures](#).

When used as icon for active objects, the picture must be designed to support a variable [number of states](#).

## JSON Grammar

Name	Data Type	Possible Values
iconPicture	Relative or filesystem path in POSIX syntax.	

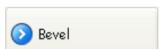
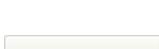
## Objects Supported

[Button](#) (all styles except [Help](#)) - [Check Box](#) - [List Box Header](#) - [Radio Button](#)

## Title/Picture Position

This property allows modifying the relative location of the button title in relation to the associated icon. This property has no effect when the button contains only a title (no associated picture) or a picture (no title). By default, when a button contains a title and a picture, the text is placed below the picture.

Here are the results using the various options for this property:

Option	Description	Example
<b>Left</b>	The text is placed to the left of the icon. The contents of the button are aligned to the right.	
<b>Top</b>	The text is placed above the icon. The contents of the button are centered.	
<b>Right</b>	The text is placed to the right of the icon. The contents of the button are aligned to the left.	
<b>Bottom</b>	The text is placed below the icon. The contents of the button are centered.	
<b>Centered</b>	The text of the icon is centered vertically and horizontally in the <b>Centered</b> button. This parameter is useful, for example, for text included in an icon.	

## JSON Grammar

Name	Data Type	Possible Values
textPlacement	string	"left", "top", "right", "bottom", "center"

## Objects Supported

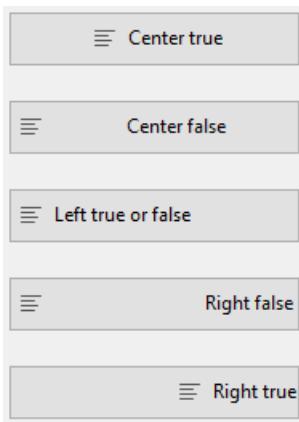
[Button](#) (all styles except [Help](#)) - [Check Box](#) - [Radio Button](#)

## Image hugs title

This property allows you to define whether the title and the picture of the button should be visually adjoined or separated, according to the [Title/Picture position](#) and [Horizontal Alignment](#) properties.

This property has no effect when the button contains only a title (no associated picture) or a picture (no title).

By default, when a button contains a title and a picture, the elements are joined. The following graphic shows the effect of the `imageHugsTitle` property (true when property is enabled) with different button alignments:



## JSON Grammar

Name	Data Type	Possible Values
imageHugsTitle	boolean	true (default), false

## Objects Supported

[Button](#) (all styles except [Help](#)) - [Check Box](#) (all styles except [Regular](#), [Flat](#), [Disclosure](#)

and Collapse/Expand) - [Radio Button](#) (all styles except Regular, Flat, Disclosure and Collapse/Expand).

## Vertical Margin

This property allows setting the size (in pixels) of the vertical margins of the button. This margin delimits the area that the button icon and title must not surpass.

This parameter is useful, for example, when the background picture contains borders.

This property works in conjunction with the [Horizontal Margin](#) property.

## JSON Grammar

Name	Data Type	Possible Values
customBorderYnumber		For use with "custom" style. Minimum: 0

## Objects Supported

[Custom Button](#) - [Custom Check Box](#) - [Custom Radio Button](#)

## With pop-up menu

This property allows displaying a symbol that appears as a triangle in the button to indicate the presence of an attached pop-up menu:



The appearance and location of this symbol depends on the button style and the current platform.

## Linked and Separated

To attach a pop-up menu symbol to a button, there are two display options available:



The actual availability of a "separated" mode depends on the style of the button and the platform.

Each option specifies the relation between the button and the attached pop-up menu:

- When the pop-up menu is **separated**, clicking on the left part of the button directly executes the current action of the button; this action can be modified using the pop-up menu accessible in the right part of the button.
- When the pop-up menu is **linked**, a simple click on the button only displays the pop-up menu. Only the selection of the action in the pop-up menu causes its execution.

### INFO

Refer to the [On Alternative Click event description](#) for more information on the handling of events in this case.

## Managing the pop-up menu

It is important to note that the "With Pop-up Menu" property only manages the graphic aspect of the button. The display of the pop-up menu and its values must be handled entirely by the developer, more particularly using `form events` and the [Dynamic pop up menu](#) and [Pop up menu](#) commands.

## JSON Grammar

Name	Data Type	Possible Values
popupPlacement	string	<ul style="list-style-type: none"><li>• "none"</li><li>• "linked"</li><li>• "separated"</li></ul>

## Objects Supported

[Toolbar Button](#) - [Bevel Button](#) - [Rounded Bevel Button](#) - [OS X Gradient Button](#) - [OS X Textured Button](#) - [Office XP Button](#) - [Circle Button](#) - [Custom](#)

## Web Area

### Access 4D methods

You can call 4D methods from the JavaScript code executed in a Web area and get values in return. To be able to call 4D methods from a Web area, you must activate the 4D methods accessibility property ("all").

This property is only available if the Web area [uses the embedded Web rendering engine](#).

When this property is on, a special JavaScript object named `$4d` is instantiated in the Web area, which you can [use to manage calls to 4D project methods](#).

#### JSON Grammar

Name	Data Type	Possible Values
methodsAccessibility	string	"none" (default), "all"

#### Objects Supported

[Web Area](#)

### Progression

Name of a Longint type variable. This variable will receive a value between 0 and 100, representing the page load completion percentage in the Web area. Automatically updated by 4D, cannot be modified manually.

As of 4D v19 R5, this variable is only updated on Windows if the Web area [uses the embedded Web rendering engine](#).

#### JSON Grammar

Name	Data Type	Possible Values
progressSource	string	Name of a Longint variable

#### Objects Supported

[Web Area](#)

### URL

A String type variable that designates the URL loaded or being loading by the associated Web area. The association between the variable and the Web area works in both directions:

- If the user assigns a new URL to the variable, this URL is automatically loaded by the Web area.
- Any browsing done within the Web area will automatically update the contents of the variable.

Schematically, this variable functions like the address area of a Web browser. You can represent it via a text area above the Web area.

## URL Variable and WA OPEN URL command

The URL variable produces the same effects as the [WA OPEN URL](#) command. The following differences should nevertheless be noted:

- For access to documents, this variable only accepts URLs that are RFC-compliant ("file://c:/My%20Doc") and not system pathnames ("c:¥MyDoc"). The [WA OPEN URL](#) command accepts both notations.
- If the URL variable contains an empty string, the Web area does not attempt to load the URL. The [WA OPEN URL](#) command generates an error in this case.
- If the URL variable does not contain a protocol (http, mailto, file, etc.), the Web area adds "http://", which is not the case for the [WA OPEN URL](#) command.
- When the Web area is not displayed in the form (when it is located on another page of the form), executing the [WA OPEN URL](#) command has no effect, whereas assigning a value to the URL variable can be used to update the current URL.

## JSON Grammar

Name	Data Type	Possible Values
urlSource	string	A URL.

## Objects Supported

### [Web Area](#)

## Use embedded Web rendering engine

This option allows choosing between two rendering engines for the Web area, depending on the specifics of your application:

- **unchecked** - `JSON value: system` (default): In this case, 4D uses the "best" engine corresponding to the system. This means that you automatically benefit from the latest advances in Web rendering, through HTML5 or JavaScript. However, you may notice some rendering differences between platforms. On Windows, 4D uses Microsoft Edge WebView2. On macOS, 4D uses the current version of WebKit (Safari).

On Windows, if Microsoft Edge WebView2 is not installed, 4D uses the embedded engine as system rendering engine. To know if it is installed in your system, look for "Microsoft Edge WebView2 Runtime" in your applications panel.

- **checked** - `JSON value: embedded`: In this case, 4D uses the Chromium Embedded Framework (CEF). Using the embedded Web engine means that Web area rendering and their functioning in your application are identical regardless of the platform used to run 4D (slight variations of pixels or differences related to network implementation may nevertheless be observed). When this option is chosen, you no longer benefit from automatic updates of the Web engine performed by the operating system; however, new versions of the engines are regularly provided through 4D.

The CEF engine has the following limitations:

- [WA SET PAGE CONTENT](#): using this command requires that at least one page is already loaded in the area (through a call to [WA OPEN URL](#) or an assignment to

the URL variable associated to the area).

- When URL drops are enabled by the `WA enable URL drop` selector of the [WA SET PREFERENCE](#) command, the first drop must be preceded by at least one call to [WA OPEN URL](#) or one assignment to the URL variable associated to the area.

## JSON Grammar

Name	Data Type	Possible Values
webEngine	string	"embedded", "system"

## Objects Supported

### [Web Area](#)

## Form Events

Form events are events that can lead to the execution of the form method and/or form object method(s). Form events allow you to control the flow of your application and to write code that is executed only when a specific event occurs.

In your code, you control the events using the `FORM Event` command, that returns the triggered event. For example:

```
//code of a button
If (FORM Event.code=On Clicked)
// do something when the button is clicked
End if
```

Every form and every active object on the form can listen to a predefined set of events, but only the events that you enabled at the form level and/or at every object level will actually occur.

## Event object

Each event is returned as an object by the `FORM Event` command. By default, it contains the following properties:

Property	Type	Description
objectName	text	Name of the object triggering the event - Not included if the event is triggered by the form
code	longint	Numeric value of the form event. Also returned by the <code>Form event code</code> command
description	text	Name of the form event (e.g. "On After Edit")

Additional properties are returned when the event occurs on specific objects. In particular:

- [list boxes](#) and [list box columns](#) return [additional properties](#) such as `columnName` or `isRowSelected`.
- [4D View Pro areas](#) return for example `sheetName` or `action` properties in the [On After Edit](#) event object.

## Events and Methods

When a form event occurs, 4D performs the following actions:

- First, it browses the objects of the form and calls the object method for any object (involved in the event) whose corresponding object event property has been selected.
- Second, it calls the form method if the corresponding form event property has been selected.

Do not assume that the object methods, if any, will be called in a particular order. The rule of thumb is that the object methods are always called before the form method. If an object is a subform, the object methods of the subform's list form are called, then the form method of the list form is called. 4D then continues to call the object methods of the parent form. In other words, when an object is a subform, 4D uses the same rule of thumb for the object and form methods within the subform object.

Except for the [On Load](#) and [On Unload](#) events (see below), if the form event property is not selected for a given event, this does not prevent calls to object methods for the objects whose same event property is selected. In other words, enabling or disabling an event at the form level has no effect on the object event properties.

The number of objects involved in an event depends on the nature of the event.

## Call Table

The following table summarizes how object and form methods are called for each event type:

Event	Object Methods	Form Method	Which Objects
On Load	Yes	Yes	All objects
On Unload	Yes	Yes	All objects
On Validate	Yes	Yes	All objects
On Clicked	Yes	Yes	Involved object only
On Double Clicked	Yes	Yes	Involved object only
On Before Keystroke	Yes	Yes	Involved object only
On After Keystroke	Yes	Yes	Involved object only
On After Edit	Yes	Yes	Involved object only
On Getting Focus	Yes	Yes	Involved object only
On Losing Focus	Yes	Yes	Involved object only
On Activate	Never	Yes	None
On Deactivate	Never	Yes	None
On Outside Call	Never	Yes	None
On Page Change	Never	Yes	None
On Begin Drag Over	Yes	Yes	Involved object only
On Drop	Yes	Yes	Involved object only
On Drag Over	Yes	Never	Involved object only
On Mouse Enter	Yes	Yes	All objects
On Mouse Move	Yes	Yes	All objects
On Mouse Leave	Yes	Yes	All objects
On Mouse Up	Yes	Never	Involved object only
On Menu Selected	Never	Yes	None
On Bound variable change	Never	Yes	None
On Data Change	Yes	Yes	Involved object only
On Plug in Area	Yes	Yes	Involved object only
On Header	Yes	Yes	All objects
On Printing Detail	Yes	Yes	All objects

<b>Event</b>	<b>Object Methods</b>	<b>YesForm Method</b>	<b>Which Objects</b>
On Printing Break	Yes	Yes	All objects
On Printing Footer	Yes	Yes	All objects
On Close Box	Never	Yes	None
On Display Detail	Yes	Yes	All objects
On Open Detail	Yes (List box)	Yes	None except List boxes
On Close Detail	Yes (List box)	Yes	None except List boxes
On Resize	Never	Yes	None
On Selection Change	Yes	Yes	Involved object only
On Load Record	Never	Yes	None
On Timer	Never	Yes	None
On Scroll	Yes	Never	Involved object only
On Before Data Entry	Yes (List box)	Never	Involved object only
On Column Moved	Yes (List box)	Never	Involved object only
On Row Moved	Yes (List box)	Never	Involved object only
On Column Resize	Yes (List box and 4D View Pro Area)	Never	Involved object only
On Header Click	Yes (List box and 4D View Pro Area)	Never	Involved object only
On Footer Click	Yes (List box)	Never	Involved object only
On After Sort	Yes (List box)	Never	Involved object only
On Long Click	Yes (Button)	Yes	Involved object only
On Alternative Click	Yes (Button and List box)	Never	Involved object only
On Expand	Yes (Hier. list and list box)	Never	Involved object only
On Collapse	Yes (Hier. list and list box)	Never	Involved object only
On Delete Action	Yes (Hier. list and list box)	Never	Involved object only
On URL Resource Loading	Yes (Web Area)	Never	Involved object only
On Begin URL Loading	Yes (Web Area)	Never	Involved object only
On URL Loading Error	Yes (Web Area)	Never	Involved object only
On URL Filtering	Yes (Web Area)	Never	Involved object only
On End URL Loading	Yes (Web Area)	Never	Involved object only
On Open External Link	Yes (Web Area)	Never	Involved object only
On Window Opening Denied	Yes (Web Area)	Never	Involved object only
On VP Range Changed	Yes (4D View Pro Area)	Never	Involved object

<b>Event</b>	<b>Object Methods</b>	<b>Form Method</b>	<b>Which Objects</b>
On VP Ready	Yes (4D View Pro Area)	Never	only Involved object
On Row Resize	Yes (4D View Pro Area)	Never	Involved object only

Always keep in mind that, for any event, the method of a form or an object is called if the corresponding event property is selected for the form or objects. The benefit of disabling events in the Design environment (using the Property List of the Form editor) is that you can reduce the number of calls to methods and therefore significantly optimize the execution speed of your forms.

**WARNING:** The [On Load](#) and [On Unload](#) events are generated for objects if they are enabled for both the objects and the form to which the objects belong. If the events are enabled for objects only, they will not occur; these two events must also be enabled at the form level.

## On Activate

Code	Can be called by	Definition
11	Form	The form's window becomes the frontmost window or the subform's container gets the focus

## Description

If the window of a form was sent to the background, this event is called when the window becomes the frontmost window.

This event applies to the form as a whole and not to a particular object. Consequently, if the `On Activate` form event property is selected, only the form will have its form method called.

In the case of a subform, this event is passed to the subform when the container gets the focus (if it has the [focusable](#) property).

## On After Edit

Code	Can be called by	Definition
45	<a href="#">4D View Pro area</a> - <a href="#">4D Write Pro area</a> - <a href="#">Combo Box</a> - <a href="#">Form</a> - <a href="#">Input</a> - <a href="#">Hierarchical List</a> - <a href="#">List Box</a> - <a href="#">List Box Column</a>	The contents of the enterable object that has the focus has just been modified

## Description

### General case

This event can be used filter the data entry in keyboard enterable objects at the lowest level.

When it is used, this event is generated after each change made to the contents of an enterable object, regardless of the action that caused the change, *i.e.*:

- Standard editing actions which modify content like paste, cut, delete or cancel;
- Dropping a value (action similar to paste);
- Any keyboard entry made by the user; in this case, the `On After Edit` event is generated after the `On Before Keystroke` and `On After Keystroke` events, if they are used.
- Any modification made using a language command that simulates a user action (*i.e.*, `POST KEY`).

Within the `On After Edit` event, text data being entered is returned by the [Get edited text](#) command.

### 4D View Pro

The object returned by the `FORM Event` command contains:

Property	Type	Description
code	longint	<code>On After Edit</code>
description	text	"On After Edit"
objectName	text	4D View Pro area name
sheetName	text	Name of the sheet of the event
action	text	"editChange", "valueChanged", "DragDropBlock", "DragFillBlock", "formulaChanged", "clipboardPasted"

Depending on the `action` property value, the [event object](#) will contain additional properties.

#### **action = editChange**

Property	Type	Description
range	object	Cell range
editingText	variant	The value from the current editor

#### **action = valueChanged**

<b>Property</b>	<b>Type</b>	<b>Description</b>
range	object	Cell range
oldValue	variant	Value of cell before change
newValue	variant	Value of cell after change

### **action = DragDropBlock**

<b>Property</b>	<b>Type</b>	<b>Description</b>
fromRange	object	Range of source cell range (being dragged)
toRange	object	Range of the destination cell range (drop location)
copy	boolean	Specifies if the source range is copied or not
insert	boolean	Specifies if the source range is inserted or not

### **action = DragFillBlock**

<b>Property</b>	<b>Type</b>	<b>Description</b>
fillRange	object	Range used for fill Value used for the fill. <ul style="list-style-type: none"> <li>• 0: Cells are filled with all data (values, formatting, and formulas)</li> </ul>
autoFillType	longint	• 1: Cells are filled with automatically sequential data <ul style="list-style-type: none"> <li>• 2: Cells are filled with formatting only</li> <li>• 3: Cells are filled with values but not formatting</li> <li>• 4: Values are removed from the cells</li> <li>• 5: Cells are filled automatically</li> </ul>
fillDirection	longint	Direction of the fill. <ul style="list-style-type: none"> <li>• 0: The cells to the left are filled</li> <li>• 1: The cells to the right are filled</li> <li>• 2: The cells above are filled</li> <li>• 3: The cells below are filled</li> </ul>

### **action = formulaChanged**

<b>Property</b>	<b>Type</b>	<b>Description</b>
range	object	Cell range
formula	text	The formula entered

### **action = clipboardPasted**

<b>Property</b>	<b>Type</b>	<b>Description</b>
range	object	Cell range Specifies what is pasted from the clipboard: <ul style="list-style-type: none"> <li>• 0: Everything is pasted (values, formatting, and formulas)</li> </ul>
pasteOption	longint	• 1: Only values are pasted <ul style="list-style-type: none"> <li>• 2: Only the formatting is pasted</li> <li>• 3: Only formulas are pasted</li> <li>• 4: Values and formatting are pasted (not formulas)</li> <li>• 5: Formulas and formatting are pasted (not values)</li> </ul>
pasteData	object	The data from the clipboard to be pasted <ul style="list-style-type: none"> <li>• "text" (text): The text from the clipboard</li> <li>• "html" (text): The HTML from the clipboard</li> </ul>

## Example

Here is an example handling an `On After Edit` event:

```
If (FORM Event.code=On After Edit)
  If (FORM Event.action="valueChanged")
    ALERT("WARNING: You are currently changing the value\
      from "+String(FORM Event.oldValue)+\
      " to "+String(FORM Event.newValue)+"!")
  End if
End if
```

The above example could generate an event object like this:

```
{
  "code":45,
  "description":"On After Edit",
  "objectName":"ViewProArea",
  "sheetname":"Sheet1",
  "action":"valueChanged",
  "range": {area:ViewProArea,ranges:[{column:1,row:2,sheet:1}]},
  "oldValue":"The quick brown fox",
  "newValue":"jumped over the lazy dog";
}
```

## On After Keystroke

Code	Can be called by	Definition
28	<a href="#">4D Write Pro area</a> - <a href="#">Combo Box</a> - Form - <a href="#">Input List Box</a> - <a href="#">List Box Column</a>	A character is about to be entered in the object that has the focus. <a href="#">Get edited text</a> returns the object's text <b>including</b> this character.

- History

## Description

The [On After Keystroke](#) event can generally be replaced by the [On After Edit](#) event (see below).

After the [On Before Keystroke](#) and [On After Keystroke](#) event properties are selected for an object, you can detect and handle the keystrokes within the object, using the [FORM event](#) command that will return [On Before Keystroke](#) and then [On After Keystroke](#) (for more information, please refer to the description of the [Get edited text](#) command).

These events are also activated by language commands that simulate a user action like [POST KEY](#).

The [On After Keystroke](#) event is not generated:

- in [list box columns](#) method except when a cell is being edited (however it is generated in any cases in the [list box](#) method),
- when user modifications are not carried out using the keyboard (paste, drag-and-drop, checkbox, drop down list, combo box). To process these events, you must use [On After Edit](#).

## Keystroke sequence

When an entry requires a sequence of keystrokes, the [On Before Keystroke](#) and [[On After Keystroke event](#)] events are generated only when the entry is fully validated by the user. The [Keystroke](#) command returns the validated character. This case mainly occurs:

- when using "dead" keys such as ^ or ~: events are generated only when the extended character is eventually entered (e.g. "ê" or ñ),
- when an IME (Input Code Editor) displays an intermediary dialog box where the user can enter a combination of characters: events are generated only when the IME dialog is validated.

## See also

[On Before Keystroke](#).

## On After Sort

Code	Can be called by	Definition
30	<a href="#">List Box</a> - <a href="#">List Box Column</a>	A standard sort has just been carried out in a list box column.

## Description

This event is generated just after a standard sort is performed (*i.e.* it is NOT generated if \$0 returns -1 in the [On Header Click](#) event). This mechanism is useful for storing the directions of the last sort performed by the user. In this event, the `Self` command returns a pointer to the variable of the sorted column header.

# On Alternative Click

## Code Can be called by

38    [Button - List Box](#) -  
      [List Box Column](#)

## Definition

- Buttons: The "arrow" area of a button is clicked
- List boxes: In a column of an object array, an ellipsis button ("alternateButton" attribute) is clicked

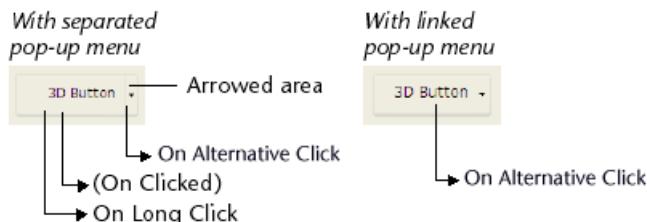
## Description

### Buttons

Some button styles can be [linked to a pop-up menu](#) and display an triangle. Clicking on this triangle causes a selection pop-up to appear that provides a set of alternative actions in relation to the primary button action.

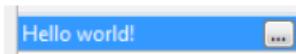
4D allows you to manage this type of button using the `On Alternative Click` event. This event is generated when the user clicks on the triangle (as soon as the mouse button is held down):

- If the pop-up menu is **separated**, the event is only generated when a click occurs on the portion of the button with the arrow. Note that the [standard action](#) assigned to the button (if any) is not executed in this case.
- If the pop-up menu is **linked**, the event is generated when a click occurs on any part of the button. Note that the [On Long Click](#) event cannot be generated with this type of button.



### List box

This event is generated in columns of [object array type list boxes](#), when the user clicks on a widget ellipsis button ("alternateButton" attribute).



See the [description of the "alternateButton" attribute](#).

## On Before Data Entry

Code	Can be called by	Definition
41	<a href="#">List Box</a> - <a href="#">List Box Column</a>	A list box cell is about to change to editing mode

### Description

This event is generated just before a cell in the list box is edited (before the entry cursor is displayed). This event allows the developer, for example, to display a different text depending on whether the user is in the display or edit mode.

When the cursor arrives in the cell, the `On Before Data Entry` event is generated in the list box or column method.

- If, in the context of this event, \$0 is set to -1, the cell is considered as not enterable. If the event was generated after **Tab** or **Shift+Tab** was pressed, the focus goes to either the next cell or the previous one, respectively.
- If \$0 is not -1 (by default \$0 is 0), the cell is enterable and switches to editing mode.

See also [Managing entry](#) section.

# On Before Keystroke

Code	Can be called by	Definition
17	<a href="#">4D Write Pro area</a> - <a href="#">Combo</a> <a href="#">Box</a> - <a href="#">Form</a> - <a href="#">Input</a> - <a href="#">List Box</a> - <a href="#">List Box Column</a>	A character is about to be entered in the object <a href="#">Box</a> - <a href="#">Form</a> - <a href="#">Input</a> - <a href="#">List Box</a> that has the focus. <a href="#">Get edited text</a> returns the object's text <b>without</b> this character.

- History

## Description

After the [On Before Keystroke](#) and [On After Keystroke event](#) events are selected for an object, you can detect and handle the keystrokes within the object, using the [Form event code](#) command that will return [On Before Keystroke](#) and then [On After Keystroke event](#) (for more information, please refer to the description of the [Get edited text](#) command). Within the [On Before Keystroke](#) event, the [FILTER KEYSTROKE](#) command can be used to filter typed chars.

These events are also activated by language commands that simulate a user action like [POST KEY](#).

The [On Before Keystroke](#) event is not generated:

- in a [list box column](#) method except when a cell is being edited (however it is generated in any cases in the [list box](#) method),
- when user modifications are not carried out using the keyboard (paste, drag-and-drop, checkbox, drop down list, combo box). To process these events, you must use [On After Edit](#).

## Non-enterable objects

The [On Before Keystroke](#) event can be generated in non-enterable objects, e.g. in a list box even if the list box cells are not enterable, or rows are not selectable. This allows you to build interfaces where the user can scroll dynamically to a specific row in a list box by entering the first letters of a value. In case where the list box cells are enterable, you can use the [Is editing text](#) command to know if the user is actually entering text in a cell or is using the type-ahead feature and then, execute appropriate code.

## Keystroke sequence

When an entry requires a sequence of keystrokes, the [On Before Keystroke](#) and [On After Keystroke](#) events are generated only when the entry is fully validated by the user. The [Keystroke](#) command returns the validated character. This case mainly occurs:

- when using "dead" keys such as ^ or ~: events are generated only when the extended character is eventually entered (e.g. "ê" or ñ),
- when an IME (Input Code Editor) displays an intermediary dialog box where the user can enter a combination of characters: events are generated only when the IME dialog is validated.

## See also

[On After Keystroke](#).



## On Begin Drag Over

Code	Can be called by	Definition
17	<a href="#">4D Write Pro area</a> - <a href="#">Button</a> - <a href="#">Button Grid</a> - <a href="#">Check Box</a> - <a href="#">Dropdown list</a> - <a href="#">Form</a> - <a href="#">Hierarchical List</a> - <a href="#">Input</a> - <a href="#">List Box</a> - <a href="#">List Box Column</a> - <a href="#">Picture</a> <a href="#">Button</a> - <a href="#">Picture Pop up menu</a> - <a href="#">Plug-in Area</a> - <a href="#">Progress Indicators</a> - <a href="#">Radio Button</a> - <a href="#">Ruler</a> - <a href="#">Spinner</a> - <a href="#">Splitter</a> - <a href="#">Stepper</a> - <a href="#">Tab control</a>	An object is being dragged

## Description

The `On Begin Drag Over` form event can be selected for any form objects that can be dragged. It is generated in every case where the object has the [Draggable](#) property. It can be called from the method of the source object or the form method of the source object.

Unlike the `On Drag Over` form event, `On Begin Drag Over` is called within the context of the **source object** of the drag action.

The `On Begin Drag Over` event is useful for preparing of the drag action. It can be used to:

- Add data and signatures to the pasteboard (via the `APPEND DATA TO PASTEBOARD` command).
- Use a custom icon during the drag action (via the `SET DRAG ICON` command).
- Accept or refuse dragging via \$0 in the method of the dragged object.
  - To indicate that drag actions are accepted, the method of the source object must return 0 (zero); you must therefore execute `$0:=0`.
  - To indicate that drag actions are refused, the method of the source object must return -1 (minus one); you must therefore execute `$0:=-1`.
  - If no result is returned, 4D considers that drag actions are accepted.

4D data are put in the pasteboard before calling the event. For example, in the case of dragging without the **Automatic Drag** action, the dragged text is already in the pasteboard when the event is called.

## On Begin URL Loading

Code Can be called by	Definition
47 <a href="#">Web Area</a>	A new URL is loaded in the Web area

## Description

This event is generated at the start of loading a new URL in the Web area. The `URL` variable associated with the Web area can be used to find out the URL being loaded.

The URL being loaded is different from the [current URL](#) (refer to the description of the `WA Get current URL` command).

## On Bound Variable Change

Code	Can be called by	Definition
54	Form	The variable bound to a subform is modified

### Description

This event is generated in the context of the form method of a [subform](#) as soon as a value is assigned to the variable bound with the subform in the parent form (even if the same value is reassigned) and if the subform belongs to the current form page or to page 0.

For more information, refer to the [Managing the bound variable](#) section.

## On Clicked

Code	Can be called by	Definition
4	<a href="#">4D View Pro Area</a> - <a href="#">4D Write Pro area</a> - <a href="#">Button</a> - <a href="#">Button Grid</a> - <a href="#">Check Box</a> - <a href="#">Combo Box</a> - <a href="#">Dropdown list</a> - <a href="#">Form</a> - <a href="#">Hierarchical List</a> - <a href="#">Input List Box</a> - <a href="#">List Box Column</a> - <a href="#">Picture Button</a> - <a href="#">Picture Pop up menu</a> - <a href="#">Plug-in Area</a> - <a href="#">Progress Indicators</a> - <a href="#">Radio Button</a> - <a href="#">Ruler</a> - <a href="#">Spinner Splitter</a> - <a href="#">Stepper</a> - <a href="#">Tab control</a>	A click occurred on an object

## Description

The `On Clicked` event is generated when the user clicks on an object.

Some form objects can be activated with the keyboard. For example, once a check box gets the focus, it can be entered using the space bar. In such a case, the `On Clicked` event is still generated.

The `On Clicked` event usually occurs once the mouse button is released. However, there are several exceptions:

- [Invisible buttons](#): The `On Clicked` event occurs as soon as the click is made and does not wait for the mouse button to be released.
- [Rulers](#): If the [Execute object method](#) option is set to **true**, the `On Clicked` event occurs as soon as the click is made.
- [Combo boxes](#): The `On Clicked` event occurs only if the user selects another value in the associated menu. A [combo box](#) must be treated as an enterable text area whose associated drop-down list provides default values. Consequently, you handle data entry within a combo box through the `On Before Keystroke`, `On After Keystroke` and `On Data Change` events.
- [Drop-down lists](#): The `On Clicked` event occurs only if the user selects another value in the menu. The `On Data Change` event allows you to detect the activation of the object when a value different from the current value is selected
- When a list box input cell is [being edited](#), the `On Clicked` event is generated when the mouse button is pressed, allowing to use the [Contextual click](#) command for example.

In the context of an `On Clicked` event, you can test the number of clicks made by the user by means of the `Clickcount` command.

## On Clicked and On Double Clicked

After the `On Clicked` or `On Double Clicked` object event property is selected for an object, you can detect and handle the clicks within or on the object, using the `FORM event` command that returns `On Clicked` or `On Double Clicked`, depending on the case.

If both events are selected for an object, the `On Clicked` and then the `On Double Clicked` events will be generated when the user double-clicks the object.

## 4D View Pro

This event is generated when the user clicks anywhere on a 4D View Pro document. On this context, the [event object](#) returned by the `FORM Event` command contains:

<b>Property</b>	<b>Type</b>	<b>Description</b>
code	longint	On Clicked
description	text	"On Clicked"
objectName	text	4D View Pro area name
sheetName	text	Name of the sheet of the event
range	object	Cell range

## **Example**

```
If (FORM Event.code=On Clicked)
    VP SET CELL STYLE(FORM Event.range;New
object("backColor";"green"))
End if
```

## On Close Box

Code Can be called by	Definition
22 Form	The window's close box has been clicked

## Description

The `On Close Box` event is generated when the user clicks on the close box of the window.

## Example

This example shows how to respond to a close window event with a form used for record data entry:

```
//Method for an input form
$vpFormTable:=Current form table
Case of
//...
:(Form event code=On Close Box)
If(Modified record($vpFormTable->))
    CONFIRM("This record has been modified. Save Changes?")
    If(OK=1)
        ACCEPT
    Else
        CANCEL
    End if
Else
    CANCEL
End if
//...
End case
```

## On Close Detail

Code Can be called by	Definition
26 Form - <a href="#">List Box</a>	You left the detail form and are going back to the output form

## Description

The `On Close Detail` event can be used in the following contexts:

- **Output forms:** the detail form is closed and the user goes back to the list form. This event cannot be selected for project forms, it is only available with **table forms**.
- List box of the [selection type](#): This event is generated when a record displayed in the [detail form](#) associated with a selection type list box is about to be closed (regardless of whether or not the record was modified).

## On Collapse

Code	Can be called by	Definition
44	- <a href="#">Hierarchical List</a>	An element of the hierarchical list or hierarchical list box has been collapsed using a click or a keystroke

## Description

- [Hierarchical list](#): This event is generated every time an element of the hierarchical list is collapsed with a mouse click or keystroke.
- [Hierarchical list boxes](#): This event is generated when a row of the hierarchical list box is collapsed.

## See also

[On Expand](#)

## On Column Moved

Code	Can be called by	Definition
32	<a href="#">List Box</a> - <a href="#">List Box Column</a>	A list box column is moved by the user via drag and drop

## Description

This event is generated when a column of the list box is moved by the user using drag and drop ([if allowed](#)). It is not generated if the column is dragged and then dropped in its initial location.

The `LISTBOX MOVED COLUMN NUMBER` command returns the new position of the column.

# On Column Resize

Code	Can be called by	Definition
33	<a href="#">4D View Pro Area - List Box</a> - <a href="#">List Box Column</a>	The width of a column is modified directly by the user or consequently to a form window resize

## Description

### List Box

This event is generated when the width of a column in the list box is modified by a user. The event is triggered "live", *i.e.*, sent continuously during the event, for as long as the list box or column concerned is being resized. This resizing is performed manually by a user, or may occur as a result of the list box and its column(s) being resized along with the form window itself (whether the form is resized manually or using the `RESIZE FORM WINDOW` command).

The `On Column Resize` event is not triggered when a [fake column](#) is resized.

### 4D View Pro

This event is generated when the width of a column is modified by a user. On this context, the [event object](#) returned by the `FORM Event` command contains:

Property	Type	Description
code	longint	On Column Resize
description	text	"On Column Resize"
objectName	text	4D View Pro area name
sheetName	text	Name of the sheet of the event
range	object	Cell range of the columns whose widths have changed
header	boolean	True if the row header column (first column) is resized, else false

### Example

```
If (FORM Event.code=On Column Resize)
    VP SET CELL STYLE(FORM Event.range;New object("hAlign";vk
horizontal align right))
End if
```

## On Data Change

Code	Can be called by	Definition
20	<a href="#">4D Write Pro area</a> - <a href="#">Dropdown list</a> - Form - <a href="#">Hierarchical List</a> - <a href="#">Input</a> - <a href="#">List Box</a> - <a href="#">List Box Column</a> - <a href="#">Plug-in Area</a> - <a href="#">Progress Indicators</a> - <a href="#">Ruler</a> - <a href="#">Spinner</a> - <a href="#">Stepper</a> - <a href="#">Subform</a>	An object data has been modified

## Description

When the `On Data Change` event property is selected for an object, you can detect and handle the change of the data source value, using the `FORM Event` command.

The event is generated as soon as the variable associated with the object is updated internally by 4D (i.e., in general, when the entry area of the object loses the focus).

With [subforms](#), the `On Data Change` event is triggered when the value of the variable of the subform object has been modified.

## On Deactivate

Code Can be called by	Definition
12 Form	The form's window ceases to be the frontmost window

## Description

If the window of a form was the frontmost window, this event is called when the window is sent to the background.

This event applies to the form as a whole and not to a particular object. Consequently, if the `On Deactivate` form event property is selected, only the form will have its form method called.

## See also

[On Activate](#)

## On Delete Action

Code	Can be called by	Definition
58	<a href="#">Hierarchical List</a> - <a href="#">List Box</a>	The user attempts to delete an item

## Description

This event is generated each time a user attempts to delete the selected item(s) by pressing a deletion key (**Delete** or **Backspace**) or selecting a menu item whose associated standard action is 'Clear' (such as the **Clear** command in the **Edit** menu).

Note that generating the event is the only action carried out by 4D: the program does not delete any items. It is up to the developer to handle the deletion and any prior warning messages that are displayed.

# On Display Detail

Code	Can be called by	Definition
8	Form - <u>List Box</u>	A record is about to be displayed in a list form or a row is about to be displayed in a list box.

## Description

The `On Display Detail` event can be used in the following contexts:

### Output form

A record is about to be displayed in a list form displayed via `DISPLAY SELECTION` and `MODIFY SELECTION`.

This event cannot be selected for project forms, it is only available with **table forms**.

In this context, the following sequence of calls to methods and form events is triggered:

- For each record:
  - For each object in the detail area:
    - Object method with `On Display Detail` event
  - Form method with `On Display Detail` event

The header area is handled using the `On Header` event.

Calling a 4D command that displays a dialog box from the `On Display Detail` event is not allowed and will cause a syntax error to occur. More particularly, the commands concerned are: `ALERT`, `DIALOG`, `CONFIRM`, `Request`, `ADD RECORD`, `MODIFY RECORD`, `DISPLAY SELECTION`, and `MODIFY SELECTION`.

### Selection list box

This event is generated when a row of a **selection type** list box is displayed.

### Displayed line number

The `Displayed line number` 4D command works with the `On Display Detail` form event. It returns the number of the row being processed while a list of records or list box rows is displayed on screen.

## On Double Clicked

Code	Can be called by	Definition
	<a href="#">4D View Pro Area</a> - <a href="#">4D Write Pro area</a> - <a href="#">Button</a> - <a href="#">Button Grid</a> - <a href="#">Check Box</a> - <a href="#">Combo Box</a> - <a href="#">Dropdown list</a> - <a href="#">Form</a> - <a href="#">Hierarchical List</a> - <a href="#">Input</a> -	A double click occurred
13	<a href="#">List Box</a> - <a href="#">List Box Column</a> - <a href="#">Picture Button</a> - <a href="#">Picture Pop up menu</a> - <a href="#">Plug-in Area</a> - <a href="#">Progress Indicators</a> - <a href="#">Radio Button</a> - <a href="#">Ruler</a> - <a href="#">Spinner</a> - <a href="#">Splitter</a> - <a href="#">Stepper</a> - <a href="#">Tab control</a>	on an object

## Description

The `On Double Clicked` event is generated when the user double-clicks on an object. The maximum length of time separating a double-click is defined in the system preferences.

If the `On Clicked` or `On Double Clicked` `onDoubleClicked.md` object event property is selected for an object, you can detect and handle the clicks within or on the object, using the `FORM event` command that returns `On Clicked` or `On Double Clicked`, depending on the case.

If both events are selected for an object, the `On Clicked` and then the `On Double Clicked` events will be generated when the user double-clicks the object.

### 4D View Pro

This event is generated when the user doubl-clicks anywhere on a 4D View Pro document. On this context, the `event object` returned by the `FORM Event` command contains:

Property	Type	Description
code	longint13	
description	text	"On Double Clicked"
objectName	text	4D View Pro area name
sheetName	text	Name of the sheet of the event
range	object	Cell range

### Example

```
If (FORM Event.code=On Double Clicked)
  $value:=VP Get value(FORM Event.range)
End if
```

## On Drag Over

Code	Can be called by	Definition
21	<a href="#">4D Write Pro area</a> - <a href="#">Button</a> - <a href="#">Button Grid</a> - <a href="#">Check Box</a> - <a href="#">Dropdown list</a> - <a href="#">Hierarchical List</a> - <a href="#">Input</a> - <a href="#">List Box</a> - <a href="#">List Box Column</a> - <a href="#">Picture Button</a> - <a href="#">Picture Pop up menu</a> - <a href="#">Plug-in Area</a> - <a href="#">Progress Indicators</a> - <a href="#">Radio</a> <a href="#">Button</a> - <a href="#">Ruler</a> - <a href="#">Spinner</a> - <a href="#">Splitter</a> - <a href="#">Stepper</a> - <a href="#">Tab control</a>	Data could be dropped onto an object

## Description

The `On Drag Over` event is repeatedly sent to the destination object when the mouse pointer is moved over the object. In response to this event, you usually:

- Get the data and signatures found in the pasteboard (via the `GET PASTEBOARD DATA` command).
- Depending on the nature and type of data in the pasteboard, you **accept** or **reject** the drag and drop.

To **accept** the drag, the destination object method must return 0 (zero), so you write `$0:=0`. To **reject** the drag, the object method must return -1 (minus one), so you write `$0:=-1`. During an `On Drag Over` event, 4D treats the object method as a function. If no result is returned, 4D assumes that the drag is accepted.

If you accept the drag, the destination object is highlighted. If you reject the drag, the destination is not highlighted. Accepting the drag does not mean that the dragged data is going to be inserted into the destination object. It only means that if the mouse button was released at this point, the destination object would accept the dragged data and the `On Drop` event would be fired.

If you do not process the `On Drag Over` event for a droppable object, that object will be highlighted for all drag over operations, no matter what the nature and type of the dragged data.

The `On Drag Over` event is the means by which you control the first phase of a drag-and-drop operation. Not only can you test whether the dragged data is of a type compatible with the destination object, and then accept or reject the drag; you can simultaneously notify the user of this fact, because 4D highlights (or not) the destination object, based on your decision.

The code handling an `On Drag Over` event should be short and execute quickly, because that event is sent repeatedly to the current destination object, due to the movements of the mouse.

## See also

[On Begin Drag Over](#)

## On Drop

Code	Can be called by	Definition
16	<a href="#">4D Write Pro area</a> - <a href="#">Button</a> - <a href="#">Button Grid</a> - <a href="#">Check Box</a> - <a href="#">Dropdown list</a> - <a href="#">Form</a> - <a href="#">Hierarchical List</a> - <a href="#">Input</a> - <a href="#">List Box</a> - <a href="#">List Box Column</a> - <a href="#">Picture</a> <a href="#">Button</a> - <a href="#">Picture Pop up menu</a> - <a href="#">Plug-in Area</a> - <a href="#">Progress Indicators</a> - <a href="#">Radio Button</a> - <a href="#">Ruler</a> - <a href="#">Spinner</a> - <a href="#">Splitter</a> - <a href="#">Stepper</a> - <a href="#">Tab control</a>	Data has been dropped onto an object

## Description

The `On Drop` event is sent once to the destination object when the mouse pointer is released over the object. This event is the second phase of the drag-and-drop operation, in which you perform an operation in response to the user action.

This event is not sent to the object if the drag was not accepted during the `On Drag Over` events. If you process the `On Drag Over` event for an object and reject a drag, the `On Drop` event does not occur. Thus, if during the `On Drag Over` event you have tested the data type compatibility between the source and destination objects and have accepted a possible drop, you do not need to re-test the data during the `On Drop`. You already know that the data is suitable for the destination object.

## See also

[On Begin Drag Over](#)

## On End URL Loading

Code Can be called by	Definition
49 <a href="#">Web Area</a>	All the resources of the URL have been loaded

### Description

This event is generated once the loading of all resources of the URL is complete. You can call the `WA Get current URL` command in order to find out the URL that was loaded.

## On Expand

Code	Can be called by	Definition
44	- <a href="#">Hierarchical List</a>	An element of the hierarchical list or hierarchical list box has been expanded using a click or a keystroke

## Description

- [Hierarchical list](#): This event is generated every time an element of the hierarchical list is expanded with a mouse click or keystroke.
- [Hierarchical list boxes](#): This event is generated when a row of the hierarchical list box is expanded.

## See also

[On Collapse](#)

## On Footer Click

Code	Can be called by	Definition
57	<a href="#">List Box</a> - <a href="#">List Box Column</a>	A click occurs in the footer of a list box column

## Description

This event is available for a list box or list box column object. It is generated when a click occurs in the footer of a list box column. In this context, the `OBJECT Get pointer` command returns a pointer to the variable of the footer that is clicked. The event is generated for both left and right clicks.

You can test the number of clicks made by the user by means of the `Clickcount` command.

## On Getting focus

Code	Can be called by	Definition
15	<a href="#">4D View Pro Area</a> - <a href="#">4D Write Pro area</a> - <a href="#">Button</a> - <a href="#">Check Box</a> - <a href="#">Combo Box</a> - <a href="#">Form</a> - <a href="#">Hierarchical List</a> - <a href="#">Input</a> - <a href="#">List Box</a> - <a href="#">List Box Column</a> - <a href="#">Plug-in Area</a> - <a href="#">Progress Indicators</a> - <a href="#">Radio Button</a> - <a href="#">Ruler</a> - <a href="#">Spinner</a> - <a href="#">Stepper</a> - <a href="#">Subform</a> - <a href="#">Web area</a>	A form object is getting the focus

## Description

The `On Getting Focus` event, along with the `On losing Focus` event, are used to detect and handle the change of focus for [focusable](#) objects.

With [subform objects](#), this event is generated in the method of the subform object when they it is checked. It is sent to the form method of the subform, which means, for example, that you can manage the display of navigation buttons in the subform according to the focus. Note that subform objects can themselves have the focus.

## On Header

Code	Can be called by	Definition
5	<a href="#">4D Write Pro area</a> - <a href="#">Button</a> - <a href="#">Button Grid</a> - <a href="#">Check Box</a> - <a href="#">Dropdown list</a> - Form (list form only) - <a href="#">Hierarchical List</a> - <a href="#">Input</a> - <a href="#">Picture Button</a> - <a href="#">Picture Pop up menu</a> - <a href="#">Plug-in Area</a> - <a href="#">Progress Indicators</a> - <a href="#">Radio Button</a> - <a href="#">Ruler</a> - <a href="#">Spinner</a> - <a href="#">Splitter</a> - <a href="#">Stepper</a> - <a href="#">Tab control</a>	The form's header area is about to be printed or displayed.

## Description

The `On Header` event is called when a record is about to be displayed in a list form displayed via `DISPLAY SELECTION` and `MODIFY SELECTION`.

This event cannot be selected for project forms, it is only available with **table forms**.

In this context, the following sequence of calls to methods and form events is triggered:

- For each object in the header area:
  - Object method with `On Header` event
  - Form method with `On Header` event

Printed records are handled using the `On Display Detail` event.

Calling a 4D command that displays a dialog box from the `On Header` event is not allowed and will cause a syntax error to occur. More particularly, the commands concerned are: `ALERT`, `DIALOG`, `CONFIRM`, `Request`, `ADD RECORD`, `MODIFY RECORD`, `DISPLAY SELECTION`, and `MODIFY SELECTION`.

## On Header Click

Code	Can be called by	Definition
42 <a href="#">4D View Pro Area</a> - <a href="#">List Box</a> - <a href="#">List Box Column</a>		A click occurs in a column header

## Description

### List Box

This event is generated when a click occurs on the header of a column in the list box. In this case, the `Self` command lets you find out the header of the column that was clicked.

If the [Sortable](#) property was selected for the list box, you can decide whether or not to authorize a standard sort of the column by passing the value 0 or -1 in the `$0` variable:

- If `$0` equals 0, a standard sort is performed.
- If `$0` equals -1, a standard sort is not performed and the header does not display the sort arrow. The developer can still generate a column sort based on customized sort criteria using the 4D language.

If the [Sortable](#) property is not selected for the list box, the `$0` variable is not used.

### 4D View Pro

This event is generated when the user clicks on a column or row header in a 4D View Pro document. In this context, the [event object](#) returned by the `FORM Event` command contains:

Property	Type	Description
code	longint	42
description	text	"On Header Click"
objectName	text	4D View Pro area name
sheetName	text	Name of the sheet of the event
range	object	Cell range The sheet location where the event took place: <ul style="list-style-type: none"><li>• 0: The crossing area between column number/letter headers (top left of the sheet)</li><li>• 1: The column headers (area indicating the column numbers/letters)</li><li>• 2: The row headers (area indicating the row numbers)</li></ul>
sheetArea	longint	

## Example

```
If (FORM Event.code=On Header Click)
Case of
  :(FORM Event.sheetArea=1)
    $values:=VP Get values(FORM Event.range)
  :(FORM Event.sheetArea=2)
    VP SET CELL STYLE(FORM Event.range;New
object("backColor";"gray"))
  :(FORM Event.sheetArea=0)
    VP SET CELL STYLE(FORM Event.range;New
object("borderBottom";\
      New object("color";"#800080";"style";vk line style thick)))
End case
End if
```

## On Load

Code	Can be called by	Definition
1	<a href="#">4D View Pro Area</a> - <a href="#">4D Write Pro area</a> - <a href="#">Button</a> - <a href="#">Button Grid</a> - <a href="#">Check Box</a> - <a href="#">Combo Box</a> - <a href="#">Dropdown list</a> - <a href="#">Form</a> - <a href="#">Hierarchical List</a> - <a href="#">Input</a> - <a href="#">List Box</a> - <a href="#">List Box Column</a> - <a href="#">Picture Button</a> - <a href="#">Picture Pop up menu</a> - <a href="#">Plug-in Area</a> - <a href="#">Progress Indicators</a> - <a href="#">Radio Button</a> - <a href="#">Ruler</a> - <a href="#">Spinner</a> - <a href="#">Splitter</a> - <a href="#">Stepper</a> - <a href="#">Subform</a> - <a href="#">Tab control</a> - <a href="#">Web Area</a>	The form is about to be displayed or printed

## Description

This event is triggered when the form is being loaded or printed.

All the objects of the form (from any page) whose `On Load` object event property is selected will have their object method called. Then, if the `On Load` form event property is selected, the form will have its form method called.

The `On Load` and `On Unload` events are generated for objects if they are enabled for both the objects and the form to which the objects belong. If the events are enabled for objects only, they will not occur; these two events must also be enabled at the form level.

## Subform

The `On Load` event is generated when opening the subform (this event must also have been activated at the parent form level in order to be taken into account). The event is generated before those of the parent form. Also note that, in accordance with the operating principles of form events, if the subform is placed on a page other than page 0 or 1, this event will only be generated when that page is displayed (and not when the form is displayed).

## See also

[On Unload](#)

## On Load Record

Code Can be called by	Definition
40 Form	During user entry in list, a record is loaded and a field is edited

## Description

The `On Load Record` event can only be used in the context of an **output form**. It is triggered during data entry in list, after a record is highlighted and a field changes to editing mode.

This event cannot be selected for project forms, it is only available with **table forms**.

## On Long Click

Code	Can be called by	Definition
39	<a href="#">Button</a>	A button is clicked and the mouse button remains pushed for a certain length of time

## Description

This event is generated when a button receives a click and the mouse button is held for a certain length of time. In theory, the length of time for which this event is generated is equal to the maximum length of time separating a double-click, as defined in the system preferences.

This event can be generated for the following button styles:

- [Toolbar](#)
- [Bevel](#)
- [Rounded Bevel](#)
- [OS X Gradient](#)
- [OS X Textured](#)
- [Office XP](#)
- [Help](#)
- [Circle](#)
- [Custom](#)

This event is generally used to display pop-up menus in case of long button clicks. The [On Clicked](#) event, if enabled, is generated if the user releases the mouse button before the "long click" time limit.

## See also

[On Alternative Click](#)

## On Losing focus

Code	Can be called by	Definition
14	<a href="#">4D View Pro Area</a> - <a href="#">4D Write Pro area</a> - <a href="#">Button</a> - <a href="#">Check Box</a> - <a href="#">Combo Box</a> - <a href="#">Form</a> - <a href="#">Hierarchical List</a> - <a href="#">Input</a> - <a href="#">List Box</a> - <a href="#">List Box Column</a> - <a href="#">Plug-in Area</a> - <a href="#">Progress Indicators</a> - <a href="#">Radio Button</a> - <a href="#">Ruler</a> - <a href="#">Spinner</a> - <a href="#">Stepper</a> - <a href="#">Subform</a> - <a href="#">Web area</a>	A form object is losing the focus

## Description

The [On Losing Focus](#) event, along with the [On Getting Focus](#) event, are used to detect and handle the change of focus for [focusable](#) objects.

With [subform objects](#), this event is generated in the method of the subform object when they it is checked. It is sent to the form method of the subform, which means, for example, that you can manage the display of navigation buttons in the subform according to the focus. Note that subform objects can themselves have the focus.

## On Menu Selected

Code Can be called by	Definition
18 Form	A menu item has been chosen in the associated menu bar

## Description

The `On Menu Selected` event is sent to the form method when a command of a menu bar associated to the form is selected. You can then call the `Menu selected` language command to test the selected menu.

You can associate a menu bar with a form in the Form properties. The menus on a form menu bar are appended to the current menu bar when the form is displayed as an output form in the Application environment.

## On Mouse Enter

Code	Can be called by	Definition
35	<a href="#">4D Write Pro area</a> - <a href="#">Button</a> - <a href="#">Button Grid</a> - <a href="#">Check Box</a> - <a href="#">Combo Box</a> - <a href="#">Dropdown list</a> - <a href="#">Form</a> - <a href="#">Hierarchical List</a> - <a href="#">Input</a> - <a href="#">List Box</a> - <a href="#">Picture Button</a> - <a href="#">Picture Pop up menu</a> - <a href="#">Plug-in Area</a> - <a href="#">Progress Indicators</a> - <a href="#">Radio Button</a> - <a href="#">Ruler</a> - <a href="#">Spinner</a> - <a href="#">Splitter</a> - <a href="#">Stepper</a> - <a href="#">Tab control</a>	The mouse cursor enters the graphic area of an object

## Description

This event is generated once, when the mouse cursor enters the graphic area of a form object.

The `On Mouse Enter` event updates the *MouseX* and *MouseY* system variables.

Objects that are made invisible using the `OBJECT SET VISIBLE` command or the [Visibility](#) property do not generate this event.

## Calling stack

If the `On Mouse Enter` event has been checked for the form, it is generated for each form object. If it is checked for an object, it is generated only for that object. When there are superimposed objects, the event is generated by the first object capable of managing it that is found going in order from top level to bottom.

## See also

- [On Mouse Move](#)
- [On Mouse Leave](#)

## On Mouse Leave

Code	Can be called by	Definition
	<a href="#">4D Write Pro area</a> - <a href="#">Button</a> - <a href="#">Button Grid</a> - <a href="#">Check Box</a> - <a href="#">Combo Box</a> - <a href="#">Dropdown list</a> - <a href="#">Form</a> - <a href="#">Hierarchical List</a> - <a href="#">Input</a> - <a href="#">List Box</a> -	The mouse cursor leaves
36	<a href="#">Picture Button</a> - <a href="#">Picture Pop up menu</a> - <a href="#">Plug-in Area</a> - <a href="#">Progress Indicators</a> - <a href="#">Radio Button</a> - <a href="#">Ruler</a> - <a href="#">Spinner</a> - <a href="#">Splitter</a> - <a href="#">Stepper</a> - <a href="#">Tab control</a>	the graphic area of an object

## Description

This event is generated once when the mouse cursor leaves the graphic area of an object.

The `On Mouse Leave` event updates the *MouseX* and *MouseY* system variables.

Objects that are made invisible using the `OBJECT SET VISIBLE` command or the [Visibility](#) property do not generate this event.

## Calling stack

If the `On Mouse Leave` event has been checked for the form, it is generated for each form object. If it is checked for an object, it is generated only for that object. When there are superimposed objects, the event is generated by the first object capable of managing it that is found going in order from top level to bottom.

## See also

- [On Mouse Move](#)
- [On Mouse Leave](#)

## On Mouse Move

Code	Can be called by	Definition
37	<a href="#">4D Write Pro area</a> - <a href="#">Button</a> - <a href="#">Button Grid</a> - <a href="#">Check Box</a> - <a href="#">Combo Box</a> - <a href="#">Dropdown list</a> - Form - <a href="#">Hierarchical List</a> - <a href="#">Input</a> - <a href="#">List Box</a> - <a href="#">Picture Button</a> - <a href="#">Picture Pop up menu</a> - <a href="#">Plug-in Area</a> - <a href="#">Progress Indicators</a> - <a href="#">Radio Button</a> - <a href="#">Ruler</a> - <a href="#">Spinner</a> - <a href="#">Splitter</a> - <a href="#">Stepper</a> - <a href="#">Tab control</a>	The mouse cursor moves at least one pixel OR a modifier key (Shift, Alt/Option, Shift Lock) was pressed

## Description

This event is generated:

- when the mouse cursor moves at least one pixel
- OR when a modifier key (**Shift, Alt/Option, Shift Lock**) was pressed. This makes it possible to manage copy- or move-type drag-and-drop operations.

If the event is checked for an object only, it is generated only when the cursor is within the graphic area of the object.

The `On Mouse Move` event updates the *MouseX* and *MouseY* system variables.

Objects that are made invisible using the `OBJECT SET VISIBLE` command or the [Visibility](#) property do not generate this event.

## Calling stack

If the `On Mouse Move` event has been checked for the form, it is generated for each form object. If it is checked for an object, it is generated only for that object. When there are superimposed objects, the event is generated by the first object capable of managing it that is found going in order from top level to bottom.

## See also

- [On Mouse Enter](#)
- [On Mouse Leave](#)

## On Mouse Up

Code	Can be called by	Definition
2	<a href="#">Input</a> of the <a href="#">picture</a> <a href="#">Type</a>	The user has just released the left mouse button in a Picture object

## Description

The `On Mouse Up` event is generated when the user has just released the left mouse button while dragging in a picture input. This event is useful, for example, when you want the user to be able to move, resize or draw objects in a SVG area.

When the `On Mouse Up` event is generated, you can get the local coordinates where the mouse button was released. These coordinates are returned in the `MouseX` and `MouseY` System variables. The coordinates are expressed in pixels with respect to the top left corner of the picture (0,0).

When using this event, you must also use the `Is waiting mouse up` command to handle cases where the "state manager" of the form is desynchronized, i.e. when the `On Mouse Up` event is not received after a click. This is the case for example when an alert dialog box is displayed above the form while the mouse button has not been released. For more information and an example of use of the `On Mouse Up` event, please refer to the description of the `Is waiting mouse up` command.

If the [Draggable](#) option is enabled for the picture object, the `On Mouse Up` event is never generated.

## On Open Detail

Code	Can be called by	Definition
25	Form - <u>List Box</u>	The detail form associated with the output form or with the list box is about to be opened.

## Description

The `On Open Detail` event can be used in the following contexts:

- **Output forms:** A record is about to be displayed in the detail form associated with the output form. This event cannot be selected for project forms, it is only available with **table forms**.
- List box of the [selection type](#): This event is generated when a record is about to be displayed in the detail form associated with a list box of the selection type (and before this form is opened).

## Displayed line number

The `Displayed line number` 4D command works with the `On Open Detail` form event. It returns the number of the row being processed while a list of records or list box rows is displayed on screen.

## On Open External Link

Code Can be called by	Definition
52 <a href="#">Web Area</a>	An external URL has been opened in the browser

### Description

This event is generated when the loading of a URL was blocked by the Web area and the URL was opened with the current system browser, because of a filter set up via the `WA SET EXTERNAL LINKS FILTERS` command.

You can find out the blocked URL using the `WA Get last filtered URL` command.

### See also

[On URL Filtering](#)

## On Outside Call

Code	Can be called by	Definition
10	Form	The form received a <code>POST OUTSIDE CALL</code> call

## Description

This event is called when the form is called from another process through the `POST OUTSIDE CALL` command.

The `On Outside Call` event modifies the entry context of the receiving input form. In particular, if a field was being edited, the [`On Data Change`](#) event is generated.

## On Page Change

Code	Can be called by	Definition
56	Form	The current page of the form is changed

## Description

This event is only available at the form level (it is called in the form method). It is generated each time the current page of the form changes (following a call to the `FORM GOTO PAGE` command or a standard navigation action).

Note that it is generated after the page is fully loaded, i.e. once all the objects it contains are initialized, including [Web areas](#).

The only exception is 4D View Pro areas, for which you need to call the [On VP Ready](#) specific event.

The `On Page Change` event is useful for executing code that requires all objects to be initialized beforehand. You can also use it to optimize the application by executing code (for example, a search) only after a specific page of the form is displayed and not just as soon as page 1 is loaded. If the user does not go to this page, the code is not executed.

## On Plug in Area

Code Can be called by	Definition
19    Form - <a href="#">Plug-in Area</a>	An external object requested its object method to be executed

## Description

The event is generated when a plug-in requested its form area to execute the associated object method.

## On Printing Break

Code	Can be called by	Definition
6	<a href="#">4D Write Pro area</a> - <a href="#">Button</a> - <a href="#">Button Grid</a> - <a href="#">Check Box</a> - <a href="#">Combo Box</a> - <a href="#">Dropdown list</a> - Form - <a href="#">Hierarchical List</a> - <a href="#">Input</a> - <a href="#">Picture Button</a> - <a href="#">Picture Pop up menu</a> - <a href="#">Plug-in Area</a> - <a href="#">Progress Indicators</a> - <a href="#">Radio Button</a> - <a href="#">Ruler</a> - <a href="#">Spinner</a> - <a href="#">Splitter</a> - <a href="#">Stepper</a> - <a href="#">Tab control</a>	One of the form's break areas is about to be printed

## Description

The `On Printing Break` event can only be used in the context of an **output form**. It is triggered each time a break area in the output form is about to be printed, so that you can evaluate the break values, for example.

This event usually follows a call to the `Subtotal` command.

This event cannot be selected for project forms, it is only available with **table forms**.

## On Printing Detail

Code	Can be called by	Definition
23	<a href="#">4D Write Pro area</a> - <a href="#">Button</a> - <a href="#">Button Grid</a> - <a href="#">Check Box</a> - <a href="#">Combo Box</a> - <a href="#">Dropdown list</a> - <a href="#">Form</a> - <a href="#">Hierarchical List</a> - <a href="#">Input</a> - <a href="#">Picture Button</a> - <a href="#">Picture Pop up menu</a> - <a href="#">Plug-in Area</a> - <a href="#">Progress Indicators</a> - <a href="#">Radio Button</a> - <a href="#">Ruler</a> - <a href="#">Spinner</a> - <a href="#">Splitter</a> - <a href="#">Stepper</a> - <a href="#">Tab control</a>	- The form's detail area is about to be printed

## Description

The `On Printing Detail` event can only be used in the context of an **output form**. It is triggered when the detail area the output form is about to be printed, for example following a call to the `Print form` command.

The `Print form` command generates only one `On Printing Detail` event for the form method.

This event cannot be selected for project forms, it is only available with **table forms**.

## On Printing Footer

Code	Can be called by	Definition
7	<a href="#">4D Write Pro area</a> - <a href="#">Button</a> - <a href="#">Button Grid</a> - <a href="#">Check Box</a> - <a href="#">Combo Box</a> - <a href="#">Dropdown list</a> - <a href="#">Form</a> - <a href="#">Hierarchical List</a> - <a href="#">Input</a> - <a href="#">Picture Button</a> - <a href="#">Picture Pop up menu</a> - <a href="#">Plug-in Area</a> - <a href="#">Progress Indicators</a> - <a href="#">Radio Button</a> - <a href="#">Ruler</a> - <a href="#">Spinner</a> - <a href="#">Splitter</a> - <a href="#">Stepper</a> - <a href="#">Tab control</a>	- The form's footer area is about to be printed

## Description

The `On Printing Footer` event can only be used in the context of an **output form**. It is triggered when the footer area the output form is about to be printed, so that you can evaluate the footer values.

This event can be triggered in the context of a `PRINT SELECTION` command.

This event cannot be selected for project forms, it is only available with **table forms**.

## On Resize

	<b>Can be Code called by</b>	<b>Definition</b>
29	Form	The form's window is resized or the subform object is resized (in this case the event is generated in the form method of the subform)

## Description

This event is called:

- when the window of the form is resized,
- in the context of subforms, when the size of the subform object in the parent form has changed. In this this case, this event is sent to the subform form method.

## On Row Moved

Code	Can be called by	Definition
34	<a href="#">List Box of the array type - List Box Column</a>	A list box row is moved by the user via drag and drop

## Description

This event is generated when a row of the list box ([array type only](#)) is moved by the user using drag and drop ([if allowed](#)). It is not generated if the row is dragged and then dropped in its initial location.

The `LISTBOX MOVED ROW NUMBER` command returns the new position of the row.

## On Row Resize

Code Can be called by	Definition
60 <a href="#">4D View Pro Area</a>	The height of a row is modified by a user with the mouse

## Description

This event is generated when the height of a row is modified by a user in a 4D View Pro document. In this context, the [event object](#) returned by the `FORM Event` command contains:

Property	Type	Description
code	longint	60
description	text	"On Row Resize"
objectName	text	4D View Pro area name
sheetName	text	Name of the sheet of the event
range	object	Cell range of the rows whose heights have changed
header	boolean	True if the column header row (first row) is resized, else false

## Example

```
If (FORM Event.code=On Row Resize)
    VP SET CELL STYLE(FORM Event.range;New object("vAlign";vk vertical
align top))
End if
```

## On Scroll

Code	Can be called by	Definition
59	<a href="#">Input</a> of the <a href="#">picture</a> <a href="#">Type</a> - <a href="#">List Box</a>	The user scrolls the contents of a picture object or list box using the mouse or keyboard.

### Description

This event can be generated in the context of a picture input or a list box.

This event is triggered after any other user event related to the scrolling action ([On Clicked](#), [On After Keystroke](#), etc.). The event is only generated in the object method (not in the form method).

The event is triggered when the scroll is the result of a user action: using the scroll bars and/or cursors, using the mouse wheel or [the keyboard](#). It is not generated when the object is scrolled due to the execution of the [OBJECT SET SCROLL POSITION](#) command.

#### Picture input

The event is generated as soon as a user scrolls a picture within the picture input (field or variable) that contains it. You can scroll the contents of a picture area when the size of the area is smaller than its contents and the [display format](#) is "Truncated (non Centered)".

#### List box

The event is generated as soon as a user scrolls the rows or columns of the list box.

## On Selection Change

Code	Can be called by	Definition
31	<a href="#">4D View Pro area</a> - <a href="#">4D Write Pro area</a> - Form - <a href="#">Hierarchical List</a> - <a href="#">Input</a> - <a href="#">List Box</a>	The selection in the object is modified

### Description

This event can be generated in different contexts.

#### 4D View Pro

The current selection of rows or columns is modified. In this context, the [event object](#) returned by the `FORM Event` command contains:

Property	Type	Description
code	longint31	
description	text	"On Selection Change"
objectName	text	4D View Pro area name
sheetName	text	Name of the sheet of the event
oldSelections	object	Cell range before change
newSelections	object	Cell range after change

#### Example

```
If (FORM Event.code=On Selection Change)
    VP SET CELL STYLE(FORM Event.oldSelections;New
object("backColor";Null))
    VP SET CELL STYLE(FORM Event.newSelections;New
object("backColor";"red"))
End if
```

#### List form

The current record or the current selection of rows is modified in a list form.

#### Hierarchical list

This event is generated every time the selection in the hierarchical list is modified after a mouse click or keystroke.

#### Input & 4D Write Pro

The text selection or the position of the cursor in the area is modified following a click or a keystroke.

#### List box

This event is generated each time the current selection of rows or columns of the list box is modified.

## On Timer

Code	Can be called by	Definition
27 Form		The number of ticks defined by the <code>SET TIMER</code> command has passed

## Description

This event is generated only if the form method contains a previous call to the `SET TIMER` command.

When the `On Timer` form event property is selected, only the form method will receive the event, no object method will be called.

## On Unload

Code	Can be called by	Definition
24	<a href="#">4D View Pro Area</a> - <a href="#">4D Write Pro area</a> - <a href="#">Button</a> - <a href="#">Button Grid</a> - <a href="#">Check Box</a> - <a href="#">Combo Box</a> - <a href="#">Dropdown list</a> - <a href="#">Form</a> - <a href="#">Hierarchical List</a> - <a href="#">Input</a> - <a href="#">List Box</a> - <a href="#">List Box Column</a> - <a href="#">Picture Button</a> - <a href="#">Picture Pop up menu</a> - <a href="#">Plug-in Area</a> - <a href="#">Progress Indicators</a> - <a href="#">Radio Button</a> - <a href="#">Ruler</a> - <a href="#">Spinner</a> - <a href="#">Splitter</a> - <a href="#">Stepper</a> - <a href="#">Subform</a> - <a href="#">Tab control</a> - <a href="#">Web Area</a>	The form is about to be exited and released

## Description

This event is triggered when the form is being exited released.

All the objects of the form (from any page) whose `On Unload` object event property is selected will have their object method called. Then, if the `On Unload` form event property is selected, the form will have its form method called.

The `On Load` and `[On Unload]` events are generated for objects if they are enabled for both the objects and the form to which the objects belong. If the events are enabled for objects only, they will not occur; these two events must also be enabled at the form level.

## Subform

The `On Unload` event is generated when the subform is closing (this event must also have been activated at the parent form level in order to be taken into account). The event is generated before those of the parent form. Also note that, in accordance with the operating principles of form events, if the subform is placed on a page other than page 0 or 1, this event will only be generated when that page is closed (and not when the form is closed).

## See also

[On Load](#)

## On URL Filtering

Code	Can be called by	Definition
51	<a href="#">Web Area</a>	A URL was blocked by the Web area

### Description

This event is generated when the loading of a URL is blocked by the Web area because of a filter set up using the `WA SET URL FILTERS` command.

You can find out the blocked URL using the `WA Get last filtered URL` command.

### See also

[On Open External Link](#)

## On URL Loading Error

Code	Can be called by	Definition
50	<a href="#">Web Area</a>	An error occurred when the URL was loading

## Description

This event is generated when an error is detected during the loading of a URL.

You can call the `WA GET LAST URL ERROR` command in order to get information about the error.

## See also

[On Open External Link](#)

## On URL Resource Loading

Code Can be called by	Definition
48 <a href="#">Web Area</a>	A new resource is loaded in the Web area

### Description

This event is generated each time a new resource (picture, frame, etc.) is loaded on the current Web page.

The [Progression](#) variable associated with the area lets you find out the current state of the loading.

### See also

[On Open External Link](#)

## On Validate

Code	Can be called by	Definition
	<a href="#">4D Write Pro area</a> - <a href="#">Button</a> - <a href="#">Button Grid</a> - <a href="#">Check Box</a> - <a href="#">Combo Box</a> - <a href="#">Dropdown list</a> - <a href="#">Form</a> - <a href="#">Hierarchical List</a> - <a href="#">Input</a> - <a href="#">List Box</a> - <a href="#">List Box</a>	The record
3	<a href="#">Column</a> - <a href="#">Picture Button</a> - <a href="#">Picture Pop up menu</a> - <a href="#">Plug-in Area</a> - <a href="#">Progress Indicators</a> - <a href="#">Radio Button</a> - <a href="#">Ruler</a> - <a href="#">Spinner</a> - <a href="#">Splitter</a> - <a href="#">Stepper</a> - <a href="#">Subform</a> - <a href="#">Tab control</a>	data entry has been validated

## Description

This event is triggered when the record data entry has been validated, for example after a `SAVE RECORD` command call or an `accept standard action`.

## Subform

The `On Validate` event is triggered when data entry is validated in the subform.

## On VP Range Changed

Code	Can be called by	Definition
61	<a href="#">4D View Pro Area</a>	The 4D View Pro cell range has changed (e.g., a formula calculation, value removed from a cell, etc.)

### Description

This event is generated when a change occurs within a cell range in the 4D View Pro document.

The object returned by the FORM Event command contains:

Property	Type	Description
objectName	text	4D View Pro area name
code	longint	On VP Range Changed
description	text	"On VP Range Changed"
sheetName	text	Name of the sheet of the event
range	object	Cell range of the change
changedCells	object	Range containing only the changed cells. It can be a combined range.
		The type of operation generating the event:
		<ul style="list-style-type: none"><li>• "clear" - A clear range value operation</li><li>• "dragDrop" - A drag and drop operation</li><li>• "dragFill" - A drag fill operation</li><li>• "evaluateFormula" - Setting a formula in a specified cell range</li><li>• "paste" - A paste operation</li><li>• "setArrayFormula" - Setting a formula in a specified cell range</li><li>• "sort" - Sorting a range of cells</li></ul>
action	text	

See also [On After Edit](#).

## On VP Ready

Code Can be called by	Definition
9 <a href="#">4D View Pro Area</a>	The loading of the 4D View Pro area is complete

### Description

This event is generated when the 4D View Pro area loading is complete.

You need to use this event to write initialization code for the area. Any 4D View Pro area initialization code, for loading or reading values from or in the area, must be located in the `On VP Ready` form event of the area. This form event is triggered once the area loading is complete. Testing this event makes you sure that the code will be executed in a valid context. An error is returned if a 4D View Pro command is called before the `On VP Ready` form event is generated.

4D View Pro areas are loaded asynchronously in 4D forms. It means that the standard [On load](#) form event cannot be used for 4D View Pro initialization code, since it could be executed before the loading of the area is complete. `On VP Ready` is always generated after [On load](#).

## On Window Opening Denied

Code Can be called by	Definition
53 <a href="#">Web Area</a>	A pop-up window has been blocked
■ History	

### Description

This event is generated when the opening of a pop-up window is blocked by the Web area. 4D Web areas do not allow the opening of pop-up windows.

You can find out the blocked URL using the `WA Get last filtered URL` command.

This event is also triggered when a drop operation has been done in the Web area (with embedded and Wwindows system [engines](#)) if the [Drag and drop](#) option is also enabled for the area. You can accept the drop by calling:

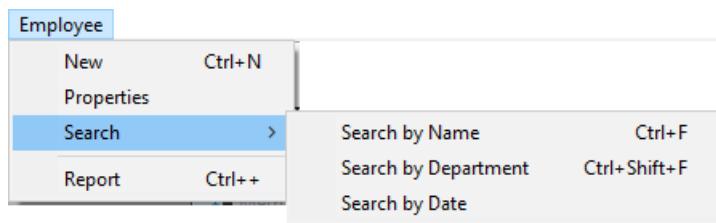
```
//web area object method
If (FORM Event.code=On Window Opening Denied)
    WA OPEN URL(*; "WebArea"; WA Get last filtered URL(*; "WebArea"))
    // or UrlVariable:=WA Get last filtered URL(*; "WebArea")
    // where UrlVariable is the URL variable associated to the web area
End if
```

### See also

[On Open External Link](#)

## Menus

You can create menu bars and menus for your 4D applications. Because pull-down menus are a standard feature of any desktop application, their addition will make your applications easier to use and will make them feel familiar to users.



A **menu bar** is a group of menus that can be displayed on a screen together. Each **menu** on a menu bar can have numerous menu commands in it, including some that call cascading submenus (or hierarchical submenus). When the user chooses a menu or submenu command, it calls a project method or a standard action that performs an operation.

You can have many separate menu bars for each application. For example, you can use one menu bar that contains menus for standard operations on the database and another that becomes active only for reporting. One menu bar may contain a menu with menu commands for entering records. The menu bar appearing with the input form may contain the same menu, but the menu commands are disabled because the user doesn't need them during data entry.

You can use the same menu in several menu bars or other menus, or you can leave it unattached and manage it only by programming (in this case, it is known as an independent menu).

When you design menus, keep the following two rules in mind:

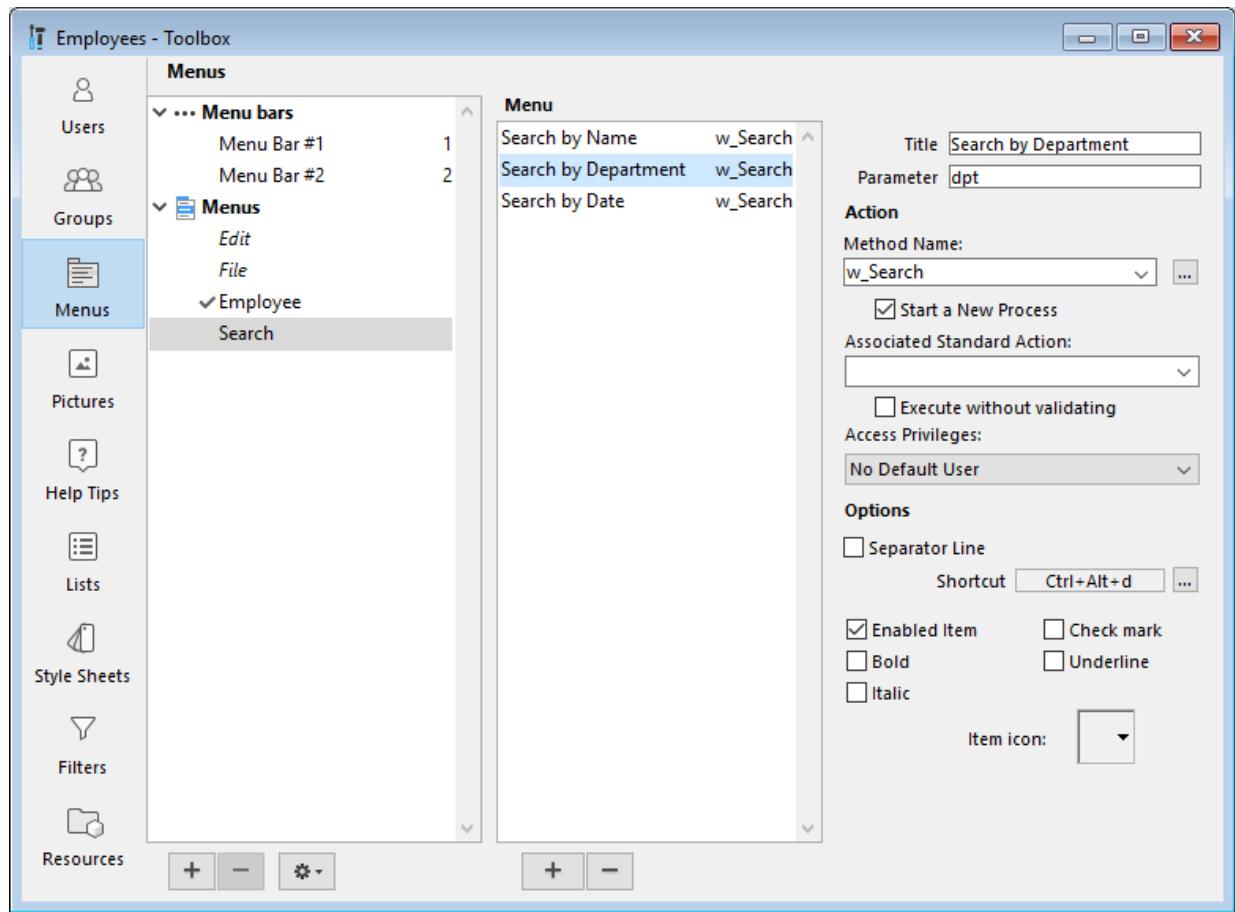
- Use menus for functions that are suited to menus: Menu commands should perform tasks such as adding a record, searching for records, or printing a report.
- Group menu commands by function: For example, all menu commands that print reports should be in the same menu. For another example, you might have all the operations for a certain table in one menu.

To create menus and menu bars, you can use either:

- the Menu editor from the Toolbox,
- language commands for the "Menus" theme,
- a combination of both.

## Menu editor

The Menu editor is accessed using the **Menus** button of the Toolbox.



Menus and menu bars are displayed as two items of the same hierarchical list, on the left side of the dialog box. Each menu can be attached to a menu bar or to another menu. In the second case, the menu becomes a sub-menu.

4D assigns menu bar numbers sequentially — Menu Bar #1 appears first. You can rename menu bars but you cannot change their numbers. These numbers are used by the language commands.

## Creating menus and menu bars

You can create menus and menu bars:

- using the Menus editor of the 4D Toolbox window. In this case, menus and menu bars are stored in the application's structure.
- dynamically, using the language commands from the "Menus" theme. In this case, menus and menu bars are not stored, they only exist in memory.

You can combine both features and use menus created in structure as templates to define menus in memory.

### Default menu bar

A custom application must contain at least one menu bar with one menu. By default, when you create a new project, 4D automatically creates a default menu bar (Menu Bar #1) so that you can access the Application environment. The default menu bar includes standard menus and a command for returning to the Design mode.

This allows the user to access the Application environment as soon as the project is created. Menu Bar #1 is called automatically when the **Test Application** command is chosen in the **Run** menu.

The default menu bar includes three menus:

- **File**: only includes the **Quit** command. The *Quit* standard action is associated with the command, which causes the application to quit.
- **Edit**: standard and completely modifiable. Editing functions such as copy, paste, etc. are defined using standard actions.
- **Mode**: contains, by default, the **Return to Design mode** command, which is used to exit the Application mode.

Menu items appear *in italics* because they consist of references and not hard-coded text. Refer to [Title property](#).

You can modify this menu bar as desired or create additional ones.

## Creating menus

### Using the Menu editor

1. Select the item you want to create and click the add  button below the menu bar area. OR Choose **Create a new menu bar** or **Create a new menu** from the context menu of the list or the options menu below the list. If you created a menu bar, a new bar appears in the list containing the default menus (File and Edit).
2. (optional) Double-click on the name of the menu bar/menu to switch it to editing mode and enter a custom name. OR Enter the custom name in the "Title" area. Menu bar names must be unique. They may contain up to 31 characters. You can enter the name as "hard coded" or enter a reference (see [information about the Title property](#)).

### Using the 4D language

Use the `Create menu` command to create a new menu bar or menu reference (`MenuRef`) in memory.

When menus are handled by means of *MenuRef* references, there is no difference per se between a menu and a menu bar. In both cases, it consists of a list of items. Only their use differs. In the case of a menu bar, each item corresponds to a menu which is itself composed of items.

Create menu can create empty menus (to fill using APPEND MENU ITEM or INSERT MENU ITEM) or by menus built upon menus designed in the Menu editor.

## Adding items

For each of the menus, you must add the commands that appear when the menu drops down. You can insert items that will be associated with methods or standard actions, or attach other menus (submenus).

### Using the Menu editor

To add a menu item:

1. In the list of source menus, select the menu to which you want to add a command. If the menu already has commands, they will be displayed in the central list. If you want to insert the new command, select the command that you want it to appear above. It is still be possible to reorder the menu subsequently using drag and drop.
2. Choose **Add an item to menu "MenuItem"** in the options menu of the editor or from the context menu (right click in the central list). OR Click on the add  button located below the central list. 4D adds a new item with the default name "Item X" where X is the number of items already created.
3. Double-click on the name of the command in order to switch it to editing mode and enter a custom name. OR Enter the custom name in the "Title" area. It may contain up to 31 characters. You can enter the name as "hard coded" or enter a reference (see below).

### Using the 4D language

Use INSERT MENU ITEM or APPEND MENU ITEM to insert or to add menu items in existing menu references.

## Deleting menus and items

### Using the Menu editor

You can delete a menu bar, a menu or a menu item in the Menu editor at any time. Note that each menu or menu bar has only one reference. When a menu is attached to different bars or different menus, any modification or deletion made to the menu is immediately carried out in all other occurrences of this menu. Deleting a menu will only delete a reference. When you delete the last reference of a menu, 4D displays an alert.

To delete a menu bar, menu or menu item:

- Select the item to be deleted and click on the delete  button located beneath the list.
- or, use the appropriate **Delete...** command from the context menu or the options menu of the editor.

It is not possible to delete Menu Bar #1.

### Using the 4D language

Use **DELETE MENU ITEM** to remove an item from a menu reference. Use **RELEASE MENU** to unload the menu reference from the memory.

## Attaching menus

Once you have created a menu, you can attach it to one or several other menus (sub-menu) or menu bar(s).

Sub-menus can be used to group together functions organized according to subject within the same menu. Sub-menus and their items can have the same attributes as the menus themselves (actions, methods, shortcuts, icons, and so on). The items of the sub-menu keep their original characteristics and properties and the functioning of the sub-menu is identical to that of a standard menu.

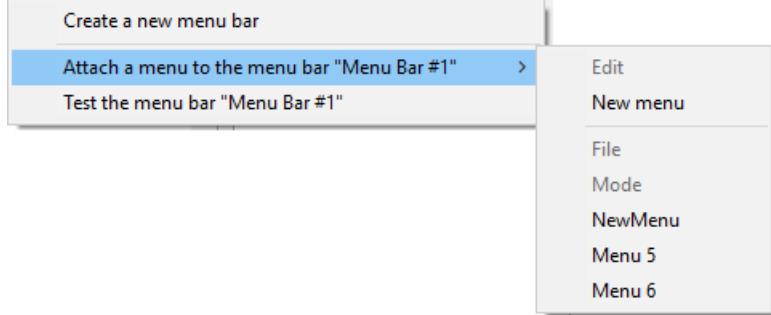
You can create sub-menus of sub-menus to a virtually unlimited depth. Note, however, that for reasons concerning interface ergonomics, it is generally not recommended to go beyond two levels of sub-menus.

At runtime, if an attached menu is modified by programming, every other instance of the menu will reflect these changes.

### Using the Menu editor

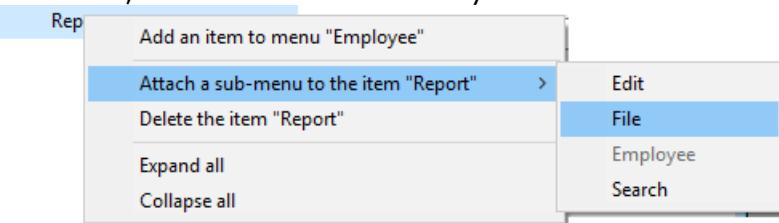
A menu can be attached to a menu bar or to another menu.

- To attach a menu to a menu bar: right-click on the menu bar and select **Attach a menu to the menu bar "bar name"**, then choose the menu to be attached to



the menu bar:  
also select a menu bar then click on the options button found below the list.

- To attach a menu to another menu: select the menu in the left-hand area, then right-click on the menu item and select **Attach a sub-menu to the item "item name"**, then choose the menu you want to use as sub-menu:



You can also select a menu item then click on the options button found below the list. The menu being attached thus becomes a sub-menu. The title of the item is kept (the original sub-menu name is ignored), but this title can be modified.

### Detaching menus

You can detach a menu from a menu bar or a sub-menu from a menu at any time. The detached menu is then no longer available in the menu bar or sub-menu as the case may be, but it is still present in the list of menus.

To detach a menu, right-click with the right button on the menu or sub-menu that you want to detach in the central list, then choose the **Detach the menu(...)** or **Detach**

**the sub-menu(...)**

## **Using the 4D language**

Since there is no difference between menus and menu bars in the 4D language, attaching menus or sub-menus is done in the same manner: use the *subMenu* parameter of the `APPEND MENU ITEM` command to attach a menu to a menu bar or an menu.

# Menu item properties

You can set various properties for menu items such as action, font style, separator lines, keyboard shortcuts or icons.

## Title

The **Title** property contains the label of a menu or menu item as it will be displayed on the application interface.

In the Menu editor, you can directly enter the label as "hard coded". Or, you can enter a reference for a variable or an XLIFF element, which will facilitate the maintenance and translation of applications. You can use the following types of references:

- An XLIFF resource reference of the type :xlfiff:MyLabel. For more information about XLIFF references, refer to *XLIFF Architecture* section in *4D Design Reference*.
- An interprocess variable name followed by a number, for example: :<>vlang, 3. Changing the contents of this variable will modify the menu label when it is displayed. In this case, the label will call an XLIFF resource. The value contained in the <>vlang variable corresponds to the *id* attribute of the *group* element. The second value (3 in this example) designates the *id* attribute of the *trans-unit* element.

Using the 4D language, you set the title property through the *itemText* parameter of the `APPEND MENU ITEM`, `INSERT MENU ITEM`, and `SET MENU ITEM` commands.

## Using control characters

You can set some properties of the menu commands by using control characters (metacharacters) directly in the menu command labels. For instance, you can assign the keyboard shortcut Ctrl+G (Windows) or Command+G (macOS) for a menu command by placing the "/G" characters in the label of the menu item label.

Control characters do not appear in the menu command labels. You should therefore avoid using them so as not to have any undesirable effects. The control characters are the following:

Character	Description	Usage
(	open parenthesis	Disable item
<B	less than B	Bold font
<I	less than I	Italic font
<U	less than U	Underline font
!+character	exclamation point+character	Add character as check mark (macOS); add check mark (Windows)
/+character	slash+character	Add character as shortcut

## Parameter

You can associate a custom parameter with each menu item. A menu item parameter is a character string whose contents can be freely chosen. It can be set in the Menu editor, or through the `SET MENU ITEM PARAMETER` command.

Menu item parameters are useful with programmed management of menus, in particular when using the `Dynamic pop up menu`, `Get menu item parameter` and

Get selected menu item parameter commands.

## Action

Each menu command can have a project method or a standard action attached to it. When the menu command is chosen, 4D executes the associated standard action or project method. For example, a **Monthly Report** menu command can call a project method that prepares a monthly report from a table containing financial data. The **Cut** menu command usually calls the `cut` standard action in order to move the selection to the clipboard and erase it from the window in the foreground.

If you do not assign a method or a standard action to a menu command, choosing that menu command causes 4D to exit the Application environment and go to the Design environment. If only the Application environment is available, this means quitting to the Desktop.

Standard actions can be used to carry out various current operations linked to system functions (copy, quit, etc.) or to those of the database (add record, select all, etc.).

You can assign both a standard action and a project method to a menu command. In this case, the standard action is never executed; however, 4D uses this action to enable/disable the menu command according to the current context and to associate a specific operation with it according to the platform. When a menu command is deactivated, the associated project method cannot be executed.

The choice between associating a standard action or a project method with a menu command depends on the type of result desired. In principle, it is preferable to choose a standard action whenever possible since they implement optimized mechanisms, more particularly activation/deactivation according to the context.

### Associating a project method or a standard action

You can assign a project method and/or a standard action to a selected menu command in the Menu editor:

- **Method Name:** Select an existing project method name in the combo box. If the project method does not exist, enter its name in the "Method Name" combo box then click on the [...] button. 4D displays a project method creation dialog that is used to access the Code Editor.
- **Associated Standard Action:** Choose or write the action you want to assign in the "Associated Standard Action" combo box. You can enter any supported action and (optionally) parameter you want in the area. For a comprehensive list of standard actions, please refer to the **Standard actions** section in the *Design Reference*. **Note for macOS:** Under macOS, the custom menu commands associated with the *Quit* action are automatically placed in the application menu, in compliance with the platform interface standards.

Using the 4D language, you can associate a project method using the `SET MENU ITEM METHOD` command, and a standard action using the `SET MENU ITEM PROPERTY` command.

### Start a new process

The **Start a New Process** option is available for menu commands associated to methods. It can be set through a check box in the Menu editor, or through the *property* parameter of the `SET MENU ITEM PROPERTY` command.

When the **Start a New Process** option is enabled, a new process is created when the

menu command is chosen. Normally, a method attached to a menu command executes within the current process unless you explicitly call a new process in your code. The **Start a New Process** option makes it easier to start a new process. When enabled, 4D will create a new process when the menu command is chosen.

In the Process list, 4D assigns the new process a default name using the format "MLProcessNumber". *The names of processes started from a menu are created by combining the prefix "ML" with the process number.*

## Execute without validating

The **Execute without validating** option is available for menu commands associated to standard actions in the Menu editor only.

When this option is checked, 4D does not trigger the "validation" of the field where the cursor is located before executing the associated action. This option is mainly intended for **Edit** menu commands. By default, 4D processes and "validates" the contents of a field before executing a standard action (via a menu command or a shortcut), which has the effect of generating an `On Data Change` form event. This can disrupt the functioning of copy or paste type commands because when they are called, the `On Data Change` form event is generated unexpectedly. In this case, it is useful to check the **Execute without validating** option.

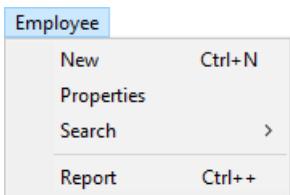
## Remote access privileges

This Menu editor option allows defining a group to a menu command so that only users in that group can use the menu command from a 4D remote application (see Users and groups).

## Options

### Separator lines

Groups of menu commands in a menu can be divided by a separator line. This convention is useful for grouping associated menu commands by function.



You add a separator line by creating a specific menu command.

In the Menu editor, instead of entering the menu command's text in the title area, you simply select the **Separator Line** option. Instead of text, a line appears in the current menu bar area. When this option is checked, the other properties have no effect.

**Note:** Under macOS, if you use the dash “-” as the first character of a menu item, it will appear as a separator line.

In the 4D language, you insert a separator line by entering `-` or `(-` as itemText for `APPEND MENU ITEM`, `INSERT MENU ITEM`, or `SET MENU ITEM` commands.

### Keyboard shortcuts

You can add keyboard shortcuts to any menu command. If a menu command has one of these keyboard shortcuts, users will see it next to the menu command. For example, "Ctrl+C" (Windows) or "Command+C" (macOS) appears next to the **Copy**

menu command in the **Edit** menu.

You can also add the **Shift** key as well as the **Alt** key (Windows) or **Option** key (macOS) to the shortcut associated with a menu command. This multiplies the number of shortcuts that can be used. The following types of keyboard shortcuts can therefore be defined:

- Under Windows:

- Ctrl+character
- Ctrl+Shift+character
- Ctrl+Alt+character
- Ctrl+Shift+Alt+character

- Under macOS:

- Command+character
- Command+Shift+character
- Command+Option+character
- Command+Shift+Option+character

We recommend that you keep the default keyboard shortcuts that are associated with standard actions.

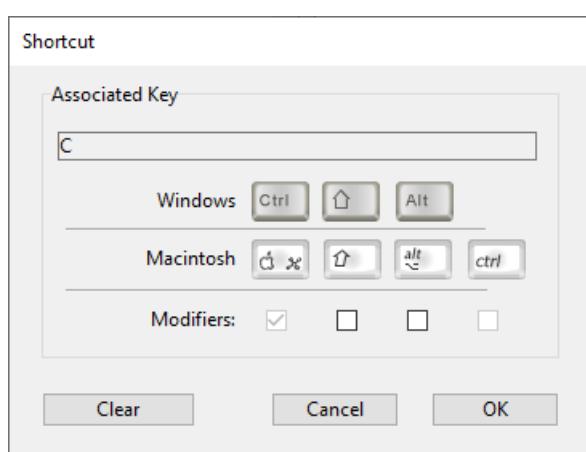
You can use any alphanumeric keys as a keyboard shortcut, except for the keys reserved by standard menu commands that appear in the **Edit** and **File** menus, and the keys reserved for 4D menu commands.

These reserved key combinations are listed in the following table:

Key (Windows)	Key (macOS)	Operation
Ctrl+C	Command+C	Copy
Ctrl+Q	Command+Q	Quit
Ctrl+V	Command+V	Paste
Ctrl+X	Command+X	Cut
Ctrl+Z	Command+Z	Undo
Ctrl+. (period)	Command+. (period)	Stop action

To assign a keyboard shortcut in the Menu editor:

Select the menu item to which you want to assign a keyboard shortcut. Click on the [...] button to the right of the "Shortcut" entry area. The following window appears:



Enter the character to use then (optional) click the **Shift** and/or **Alt (Option)** checkboxes according to the combination desired. You can also directly press the keys

that make up the desired combination (do not press the **Ctrl/Command** key).

You cannot deselect the Ctrl/Command key, which is mandatory for keyboard shortcuts for menus. To start over, click on **Clear**. Click **OK** to validate the changes. The shortcut defined is shown in the "Shortcut" entry area.

To assign a keyboard shortcut using the 4D language, use the `SET ITEM SHORTCUT` command.

An active object can also have a keyboard shortcut. If the **Ctrl/Command** key assignments conflict, the active object takes precedence.

### Enabled item

In the Menu editor, you can specify whether a menu item will appear enabled or disabled. An enabled menu command can be chosen by the user; a disabled menu command is dimmed and cannot be chosen. When the **Enabled Item** check box is unchecked, the menu command appears dimmed, indicating that it cannot be chosen.

Unless you specify otherwise, 4D automatically enables each menu item you add to a custom menu. You can disable an item in order, for example, to enable it only using programming with `ENABLE MENU ITEM` and `DISABLE MENU ITEM` commands.

### Check mark

This Menu editor option can be used to associate a system check mark with a menu item. You can then manage the display of the check mark using language commands (`SET MENU ITEM MARK` and `Get menu item mark`).

Check marks are generally used for continuous action menu items and indicate that the action is currently underway.

### Font styles

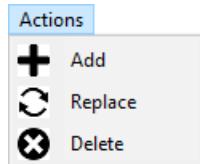
4D lets you customize menus by applying different font styles to the menu commands. You can customize your menus with the Bold, Italic or Underline styles through options in the Menu editor, or using the `SET MENU ITEM STYLE` language command.

As a general rule, apply font styles sparingly to your menus — too many styles will be distracting to the user and give a cluttered look to your application.

You can also apply styles by inserting special characters in the menu title (see [Using control characters](#) above).

### Item icon

You can associate an icon with a menu item. It will displayed directly in the menu, next to the item:



To define the icon in the Menu editor, click on the "Item icon" area and select **Open** to open a picture from the disk. If you select a picture file that is not already stored in the project resources folder, it is automatically copied in that folder. Once set, the item icon appears in the preview area:



To remove the icon from the item, choose the **No Icon** option from the "Item Icon" area.

To define item icons using the 4D language, call the `SET MENU ITEM ICON` command.

## Menu bar features

Menu bars provide the major interface for custom applications. For each custom application, you must create at least one menu bar with at least one menu. By default, Menu Bar #1 is the menu bar displayed in the Application environment. You can change which menu bar is displayed using the `SET MENU BAR` command.

4D lets you associate a custom splash screen picture with each menu bar and to preview this menu bar at any time.

## Splash screen

You can enhance the appearance of each menu bar by associating a custom splash screen with it. The window containing the splash screen is displayed below the menu bar when it appears. It can contain a logo or any type of picture. By default, 4D displays the 4D logo in the splash screen:

A custom splash screen picture can come from any graphic application. 4D lets you paste a clipboard picture or use any picture present on your hard disk. Any standard picture format supported by 4D can be used.

The splash screen picture can be set only in the Menu editor: select the menu bar with which you want to associate the custom splash screen. Note the "Background Image" area in the right-hand part of the window. To open a picture stored on your disk directly, click on the **Open** button or click in the "Background Image" area. A pop-up menu appears:

- To paste a picture from the clipboard, choose **Paste**.
- To open a picture stored in a disk file, choose **Open**. If you choose Open, a standard Open file dialog box will appear so that you can select the picture file to be used. Once set, the picture is displayed in miniature in the area. It is then associated with the menu bar.

You can view the final result by testing the menu bar (see the following section). In Application mode, the picture is displayed in the splash screen with the "Truncated (Centered)" type format.

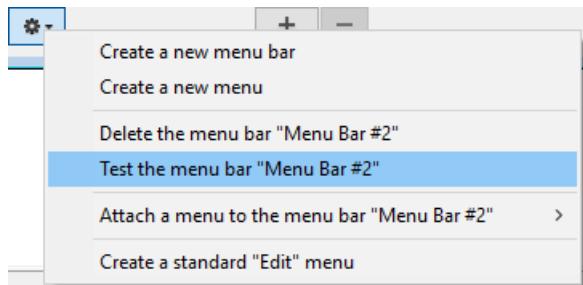
You can choose whether to display or hide this window using the **Display toolbar** option in the Settings.

To remove the custom picture and display the default one instead, click on the **Clear** button or select **Clear** in the area pop-up menu.

## Previewing menu bars

The Menu Bar editor lets you view the custom menus and splash screen at any time, without closing the toolbox window.

To do so, simply select the menu bar and choose **Test the menu bar "Menu Bar #X"** in the context menu or the options menu of the editor.



4D displays a preview of the menu bar as well as the splash screen. You can scroll down the menus and sub-menus to preview their contents. However, these menus are not active. To test the functioning of menus and the toolbar, you must use the **Test Application** command from the **Run** menu.

 INFO

If the **Use SDI mode on Windows** option is selected in the ["Interface" page of the Settings dialog box](#), the **Test Application** menu allows you to test your application in [SDI or MDI mode](#) on Windows:

## SDI mode on Windows

On Windows, 4D developers can test and configure their 4D merged applications to work as SDI (Single-Document Interface) applications. In SDI applications, each window is independent from others and can have its own menu bar. SDI applications are opposed to MDI (Multiple Documents Interface) applications, where all windows are contained in and depend on the main window.



The concept of SDI/MDI does not exist on macOS. This feature concerns Windows applications only and related options are ignored on macOS.

## SDI mode availability

The SDI mode is available in the following execution environments only:

- Windows
- Merged [stand-alone](#) or [client](#) 4D application
- [Test application feature](#) available from the **Run** menu.

## Enabling the SDI mode

To enable the SDI mode in your application, just check the **Use SDI mode on Windows** option in the ["Interface" page of the Settings dialog box](#).

Once enabled, to actually run your application in SDI mode, you can either:

- build a merged application (standalone and/or client application) and execute it on Windows, or
- select **Test Application in SDI Mode** from the **Run** menu on Windows to test the development.



Because the development environment is executed in MDI, switching from development mode to runtime mode using the **Test Application in SDI Mode** menu item is equivalent to restarting your application.

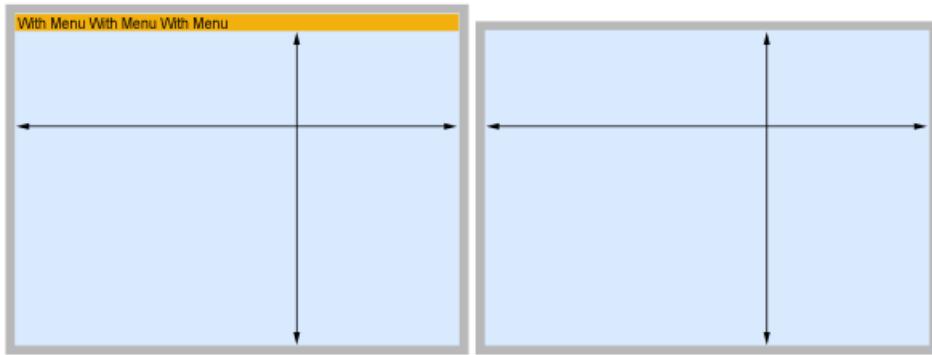
## Managing applications in SDI mode

Executing a 4D application in SDI mode does not require any specific implementation: existing menu bars are automatically moved in SDI windows themselves. However, you need to pay attention to specific principles that are listed below.

### Menus in Windows

In SDI mode, the process menu bar is automatically displayed in every document type window opened during the process life (this excludes, for example, floating palettes). When the process menu bar is not visible, menu item shortcuts remain active however.

Menus are added above windows without modifying their contents size:



Windows can therefore be used in MDI or SDI modes without having to recalculate the position of objects.

## Splash screen

- If the **Splash screen** interface option was [selected in the Settings](#), the splash window will contain any menus that would have been displayed in the MDI window. Note also that closing the splash screen window will result in exiting the application, just like in MDI mode.
- If the Splash screen option was not selected, menus will be displayed in opened windows only, depending on the programmer's choices.

## Debugger

When displayed in SDI mode, the [debugger window](#) does not contain [editing buttons](#), because switching to development environment requires to abort execution and restart the application in MDI mode.

## Automatic quit

When executed in MDI mode, a 4D application simply quits when the user closes the application window (MDI window). However, when executed in SDI mode, 4D applications do not have an application window and, on the other hand, closing the last opened window does not necessarily mean that the user wants the application to quit (faceless processes can be running, for example) -- although it could be what they want.

To handle this case, 4D applications executed in SDI mode include a mechanism to automatically quit (by calling the `QUIT 4D` command) when the following conditions are met:

- the user cannot interact anymore with the application
- there are no live user processes
- 4D processes or worker processes are waiting for an event
- the Web server is not started
- the [WebAdmin server](#) is not started.

### NOTE

When a menu with an associated *quit* standard action is called, the application quits and all windows are closed, wherever the menu was called from.

## Language

Although it is transparently handled by 4D, the SDI mode introduces small variations in the application interface management. Specificities in the 4D language are listed

below.

Command/feature	Specificity in SDI mode on Windows
Open form window	Options to support floating windows in SDI (Controller form window) and to remove the menu bar (Form has no menu bar)
Menu bar height	Returns the height in pixels of a single menu bar line even if the menu bar has been wrapped on two or more lines. Returns 0 when the command is called from a process without a form window
SHOW MENU BAR / HIDE MENU BAR	Applied to the current form window only (from where the code is executed)
MAXIMIZE WINDOW	The window is maximized to the screen size
CONVERT COORDINATES	<code>XY Screen</code> is the global coordinate system where the main screen is positioned at (0,0). Screens on its left side or on top of it can have negative coordinates and any screens on its right side or underneath it can have coordinates greater than the values returned by <code>Screen height</code> or <code>Screen width</code> .
GET MOUSE	Global coordinates are relative to the screen
GET WINDOW RECT	When -1 is passed in window parameter, the command returns 0;0;0;0
On Drop database method	Not supported
ⓘ INFO	

You can use the [Get application info](#) command to know the current running mode on Windows.

## User Settings

4D provides two modes of operation for project Settings:

- **Standard** mode: all [settings](#) are stored in the [settings.4DSettings file at the project level](#) and are applied in all cases. This is the default mode, suitable for development phase (all applications).
- **User settings** mode: part of the custom settings are stored in a [settings.4DSettings file in the Settings folder](#) (for all data files) or [in the Data folder](#) (for this data file) and are used instead of the structure settings. This mode is suitable for deployment phase for Desktop applications. You enable this mode using an option located on the [Security page](#) of the Settings.

By defining user settings, you can keep custom settings between updates of your 4D applications, or manage different settings for the same 4D application deployed on several different sites. It also makes it possible to use programming to manage setting files using XML.

4D can generate and use two types of user settings:

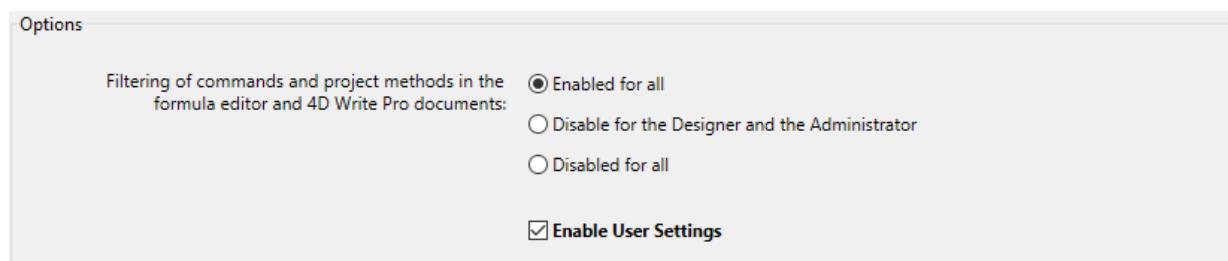
- **User Settings**: They are used instead of structure settings for any data file opened with the application.
- **User Settings for Data file**: They can be defined specifically for each data file used with your application, configuring for example the port ID or the server cache.

With this option, you can easily deploy and update several copies of the same desktop application with several data files, each containing different settings.

Consider for example the following configuration, where an application is duplicated and each copy uses a different Port ID setting. If this user setting is linked to the data file, you will be able to update the application without having to manually change the Port ID:

## Enabling User settings

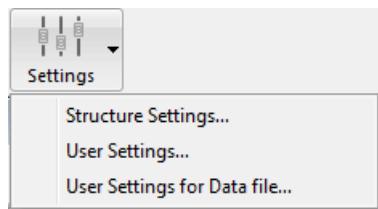
To enable user settings, you need to check the **Settings > Security > Enable User Settings** option:



When you check this option, the settings are separated into three dialog boxes:

- **Structure Settings**
- **User Settings**
- **User Settings for Data file**

You can access these dialog boxes using the **Design > Settings...** menu or the **Settings** button in the toolbar:



You can also access these dialog boxes using the [OPEN SETTINGS WINDOW](#) command with the appropriate *settingsType* selector.

The Structure Settings dialog box is identical to the standard Settings, and provides access to all its properties (which can be overridden by user settings).

## User Settings and User Settings for Data file

The **User Settings** and **User Settings for Data File** dialog boxes contain a selection of relevant properties that can be defined for all data files or a single data file:

The following table lists the pages of settings found in the **User Settings** and **User Settings for Data File** dialog boxes and describes their main differences with respect to standard settings:

<b>Page of Structure Settings</b>	<b>Page of User Settings</b>	<b>Page of User Settings for Data File</b>
<a href="#">General page</a>	N/a	N/a
<a href="#">Interface page</a>	Identical to standard settings	Identical to standard settings
<a href="#">Compiler page</a>	N/a	N/a
<a href="#">Database/Data storage page</a>	N/a	N/a
<a href="#">Database/Memory page</a>	Identical to standard settings	Identical to standard settings
<a href="#">Backup/Scheduler page</a>	N/a	Identical to standard settings
<a href="#">Backup/Configuration page</a>	N/a	Identical to standard settings
<a href="#">Backup/Backup &amp; Restore page</a>	N/a	Identical to standard settings
<a href="#">Client-server/Network options page</a>	Identical to standard settings	Identical to standard settings
<a href="#">Client-server/IP configuration page</a>	Identical to standard settings	Identical to standard settings
<a href="#">Web/Configuration page</a>	Identical to standard settings	Identical to standard settings
<a href="#">Web/Options (I) page</a>	Identical to standard settings	Identical to standard settings
<a href="#">Web/Options (II) page</a>	Identical to standard settings	Identical to standard settings
<a href="#">Web/Log (type) page</a>	Identical to standard settings	Identical to standard settings
<a href="#">Web/Log (backup) page</a>	Identical to standard settings	Identical to standard settings
<a href="#">Web/Web Services page</a>	Method prefixing option not available	Method prefixing option not available
<a href="#">SQL page</a>	Identical to standard settings	Identical to standard settings
<a href="#">PHP page</a>	Identical to standard settings	Identical to standard settings
<a href="#">Security page</a>	N/a	N/a
<a href="#">Compatibility page</a>	N/a	N/a

When you edit settings in this dialog box, they are automatically stored in the corresponding *settings.4DSettings* file (see below).

## **SET DATABASE PARAMETER and user settings**

Some of the user settings are also available through the [SET DATABASE PARAMETER](#) command. User settings are parameters with the **Kept between two sessions** property set to **Yes**.

When the **User Settings** feature is enabled, user settings edited by the [SET DATABASE PARAMETER](#) command are automatically saved in the user settings for the data file.

`Table sequence number` is an exception; this setting value is always saved in the data file itself.

## **settings.4DSettings files**

When you [check the \*\*Enable User Settings\*\* option](#), user settings files are automatically created. Their location depends on the type of user settings defined.

## User Settings

The standard user settings file is automatically created and placed in a settings folder at the following location:

[ProjectFolder/Settings/settings.4DSettings](#)

... where *ProjectFolder* is the name of the folder containing the project structure file.

In merged applications, the user settings file is placed at the following location:

- In single-user versions: ProjectFolder/Database/Settings/settings.4DSettings
- In client/server versions: ProjectFolder/Server  
Database/Settings/settings.4DSettings

## User Settings for Data File

The user settings file linked to the data file is automatically created and placed in a settings folder at the following location:

[Data/Settings/settings.4DSettings](#)

... where *Data* is the name of the folder containing the current data file of the application.

When the data file is located at the same level as the project structure file, structure-based and data-based user settings files share the same location and file. The **User Settings for Data File...** menu command is not proposed.

### NOTE

Settings files are XML files; they can be read and modified using integrated 4D XML commands or using an XML editor. This means that you can manage settings by programming, particularly in the context of applications compiled and merged with 4D Volume Desktop. When you modify this file by programming, the changes are only taken into account the next time the database is opened.

## Priority of settings

Settings can be stored at three levels. Each setting defined at one level overrides the same setting defined at a previous level, if any:

Priority level	Name	Location	Comments
3 (lowest)	Structure settings (or Settings when "User settings" feature not enabled)	<b><i>settings.4DSettings</i></b> file in the Sources folder (project databases) or in the Settings folder at the same level as the structure file (binary databases)	Unique location when user settings are not enabled. Applied to all copies of the application.
2	User settings (all data files)	<b><i>settings.4DSettings</i></b> file in the Settings folder at the same level as the Project folder	Overrides Structure settings. Stored within the application package.
1 (highest)	User settings (current data file)	<b><i>settings.4DSettings</i></b> file in the Settings folder at the same level as the data file	Overrides Structure settings and User settings. Applied only when the linked data file is used with the application.

Keep in mind that user settings files only contain a subset of relevant settings, while the structure file contains all custom settings, including core settings.

# Build Application

4D includes an application builder to create a project package (final build). This builder simplifies the finalization and deployment process for 4D compiled applications. It automatically handles the specific features of different operating systems and facilitates the deployment of client-server applications.

The application builder allows you to:

- Build a compiled structure, without interpreted code,
- Build a stand-alone, double-clickable application, *i.e.*, merged with 4D Volume Desktop, the 4D database engine,
- Build different applications from the same compiled structure via an XML project,
- Build homogeneous client-server applications,
- Build client-server applications with automatic updating of client and server parts.
- Save your build settings for future use (*Save settings* button).

Compiled applications are based upon [.4dz files](#) that are **read-only**. Keep in mind that using commands or functions that modify the source files (such as `CREATE INDEX` or `CREATE TABLE` (SQL)) is not possible by default in compiled applications. However, you can build specific applications that support local modifications by using the `PackProject` XML key (see [doc.4d.com](#)).

## Overview

Building a project package can be carried out using:

- either the [BUILD APPLICATION](#) command,
- or the [Build Application dialog](#).

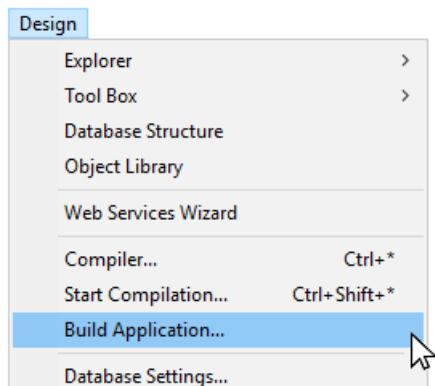


TIP

You can also download and use [Build4D](#), a component that provides classes to compile, build, and sign 4D projects, even from a headless application.

## Build application dialog

To display the Build application dialog, select **Design > Build Application...** from the menu bar.



The Build Application dialog includes several pages that can be accessed using tabs:

Building can only be carried out once the project is compiled. If you select this command without having previously compiled the project, or if the compiled code does not correspond to the interpreted code, a warning dialog box appears indicating that the project must be (re)compiled.

## buildApp.4DSettings

Each build application parameter is stored as an XML key in the application project file named `buildApp.4DSettings` XML file, located in the [Settings folder of the project](#).

Default parameters are used the first time the Build Application dialog box is used. The contents of the project file are updated, if necessary, when you click **Build** or **Save settings**. You can define several other XML settings file for the same project and employ them using the [BUILD APPLICATION](#) command.

XML keys provide additional options besides those displayed in the Build Application dialog box. The description of these keys are detailed in the [4D XML Keys BuildApplication](#) manual.

## Log file

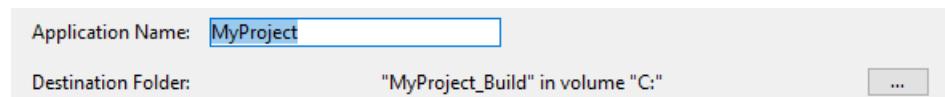
When an application is built, 4D generates a log file named `BuildApp.log.xml` in the **Logs** folder of the project. The log file stores the following information for each build:

- The start and end of building of targets,
- The name and full access path of the files generated,
- The date and time of the build,
- Any errors that occurred,
- Any signing issues (e.g. a non-signed plug-in).

Checking this file may help you saving time during the subsequent deployment steps, for example if you intend to notarize your application.

Use the `Get 4D file(Build application log file)` command to get the log file location.

## Application name and destination folder



Enter the name of the application in **Application Name**.

Specify the folder for the built application in **Destination Folder**. If the specified folder does not already exist, 4D will create a *Build* folder for you.

## Compiled structure page

This tab allows you to build a standard compiled structure file and/or a compiled component:

### Build compiled structure

Builds an application containing only compiled code.

This feature creates a **.4dz** file within a `Compiled Database/<project name>` folder. For example, if you have named your application "MyProject", 4D will create:

```
<destination>/Compiled Database/MyProject/MyProject.4dz
```

A **.4dz** file is essentially a zipped (packed) version of the project folder. **.4dz** files can be used by 4D Server, 4D Volume license (merged applications), and 4D. The compact and optimized size of **.4dz** files makes project packages easy to deploy.

When generating **.4dz** files, 4D uses a **standard** zip format by default. The advantage of this format is that it is easily readable by any unzip tool. If you do not want to use this standard format, add the `UseStandardZipFormat` XML key with value `False` in your `buildApp.4DSettings` file (for more information, see the [4D XML Keys BuildApplication](#) manual).

## Include related folders

When you check this option, any folders related to the project are copied into the Build folder as *Components* and *Resources* folders. For more information about these folders, refer to the [description of project architecture](#).

## Build component

Builds a compiled component from the structure.

A component is a standard 4D project in which specific functionalities have been developed. Once the component has been configured and installed in another 4D project (the host application project), its functionalities are accessible from the host project.

If you have named your application, *MyComponent*, 4D will create a *Components* folder containing *MyComponent.4dbase* folder:

```
<destination>/Components/MyComponent.4dbase/MyComponent.4DZ.
```

The *MyComponent.4dbase* folder contains:

- *MyComponent.4DZ* file
- A *Resources* folder - any associated Resources are automatically copied into this folder. Any other components and/or plugins folders are not copied (a component cannot use plug-ins or other components).

## Application page

This tab allows you can build a stand-alone, single-user version of your application:

### Build stand-alone Application

Checking the **Build stand-alone Application** option and clicking **Build** will create a stand-alone (double-clickable) application directly from your application project.

The following elements are required for the build:

- 4D Volume Desktop (the 4D database engine),
- an [appropriate license](#)

On Windows, this feature creates an executable file (.exe). On macOS, it handles the

creation of software packages.

The principle consists of merging a compiled structure file with 4D Volume Desktop. The functionality provided by the 4D Volume Desktop file is linked with the product offer to which you have subscribed. For more information about this point, refer to the sales documentation and to the [4D Store](#).

You can define a default data file or allow users to create and use their own data file (see the [Data file management in final applications](#) section).

It is possible to automate the update of merged single-user applications by means of a sequence of language commands (see [Automatic updating of server or single-user applications](#)).

## 4D Volume Desktop Location

In order to build a stand-alone application, you must first designate the folder containing the 4D Volume Desktop file:

- *Windows* - the folder contains the 4D Volume Desktop.4DE, 4D Volume Desktop.RSR, as well as various files and folders required for its operation. These items must be placed at the same level as the selected folder.
- *macOS* - 4D Volume Desktop is provided in the form of a structured software package containing various generic files and folders.

To select the 4D Volume Desktop folder, click on the [...] button. A dialog box appears allowing you to designate the 4D Volume Desktop folder (Windows) or package (macOS).

Once the folder is selected, its complete pathname is displayed and, if it actually contains 4D Volume Desktop, the option for building an executable application is activated.

The 4D Volume Desktop version number must match the 4D Developer Edition version number. For example, if you use 4D Developer v18, you must select a 4D Volume Desktop v18.

## Data linking mode

This option lets you choose the linking mode between the merged application and the local data file. Two data linking modes are available:

- **By application name** (default) - The 4D application automatically opens the most recently opened data file corresponding to the structure file. This allows you to move the application package freely on the disk. This option should generally be used for merged applications, unless you specifically need to duplicate your application.
- **By application path** - The merged 4D application will parse the application's *lastDataPath.xml* file and try to open the data file with an "executablePath" attribute that matches the application's full path. If such an entry is found, its corresponding data file (defined through its "dataFilePath" attribute) is opened. Otherwise, the last opened data file is opened (default mode).

For more information about the data linking mode, refer to the [Last data file opened](#) section.

## Generated files

When you click on the **Build** button, 4D automatically creates a **Final Application** folder in the specified **Destination Folder**. Inside the Final Application folder is a subfolder with the name of the specified application in it.

If you have specified "MyProject" as the name of the application, you will find the following files in this subfolder (aka MyProject):

- *Windows*

- MyProject.exe - Your executable and a MyProject.rsr (the application resources)
- 4D Extensions folder, Resources folder, various libraries (DLL), Native Components folder, SASL Plugins folder - Files necessary for the operation of the application
- Database folder - Includes a Resources folder and MyProject.4DZ file. They make up the compiled structure of the project as well as the project Resources folder. **Note:** This folder also contains the *Default Data* folder, if it has been defined (see [Data file management in final applications](#)).
- (Optional) Components folder and/or Plugins folder - Contains any components and/or plug-in files included in the project. For more information about this, refer to the [Plugins and components](#) section.
- Licenses folder - An XML file of license numbers integrated into the application. For more information about this, refer to the [Licenses & Certificate](#) section.
- Additional items added to the 4D Volume Desktop folder, if any (see [Customizing the 4D Volume Desktop folder](#)).

All these items must be kept in the same folder in order for the executable to operate.

- *macOS*

- A software package named MyProject.app containing your application and all the items necessary for its operation, including the plug-ins, components and licenses. For more information about integrating plug-ins and components, refer to the [Plugins and components](#) section. For more information about integrating licenses, refer to the [Licenses & Certificate](#) section. **Note:** In macOS, the [Application file](#) command of the 4D language returns the pathname of the ApplicationName file (located in the Contents:macOS folder of the software package) and not that of the .comp file (Contents:Resources folder of the software package).

## **Customizing 4D Volume Desktop folder**

When building a stand-alone application, 4D copies the contents of the 4D Volume Desktop folder into Destination folder > *Final Application* folder. You're then able to customize the contents of the original 4D Volume Desktop folder according to your needs. You can, for example:

- Install a 4D Volume Desktop version corresponding to a specific language;
- Add a custom *PlugIns* folder;
- Customize the contents of the *Resources* folder.

In macOS, 4D Volume Desktop is provided in the form of a software package.

In order to modify it, you must first display its contents (**Control+click** on the icon).

## Location of Web files

If your stand-alone application is used as a Web server, the files and folders required by the server must be installed in specific locations. These items are the following:

- *cert.pem* and *key.pem* files (optional): These files are used for TLS connections and by data encryption commands,
- default Web root folder.

Items must be installed:

- **on Windows:** in the *Final Application\MyProject\Database* subfolder.
- **on macOS:** next to the *MyProject.app* software package.

## Client/Server page

On this tab, you can build customized client-server applications that are homogenous, cross-platform and with an automatic update option.

### What is a Client/Server application?

A client/server application comes from the combination of three items:

- A compiled 4D project,
- The 4D Server application,
- The 4D Volume Desktop application (macOS and/or Windows).

Once built, a client/server application is composed of two customized parts: the Server portion (unique) and the Client portion (to install on each client machine).

If you want to deploy a client/server application in a heterogeneous environment (client applications running on Intel/AMD and Apple Silicon machines), it is recommended to [compile the project for all processors](#) on a macOS machine, so that all client applications will run natively.

Also, the client/server application is customized and its handling simplified:

- To launch the server portion, the user simply double-clicks on the server application. The project file does not need to be selected.
- To launch the client portion, the user simply double-clicks the client application, which connects directly to the server application. You do not need to choose a server in a connection dialog box. The client targets the server either using its name, when the client and server are on the same sub-network, or using its IP address, which is set using the `IPAddress` XML key in the `buildapp.4DSettings` file. If the connection fails, [specific alternative mechanisms can be implemented](#). You can "force" the display of the standard connection dialog box by holding down the **Option** (macOS) or **Alt** (Windows) key while launching the client application. Only the client portion can connect to the corresponding server portion. If a user tries to connect to the server portion using a standard 4D application, an error message is returned and connection is impossible.
- A client/server application can be set so that the client portion [can be updated automatically over the network](#). You only need to create and distribute an initial version of the client application, subsequent updates are handled using the automatic update mechanism.

- It is also possible to automate the update of the server part through the use of a sequence of language commands ([SET UPDATE FOLDER](#) and [RESTART 4D](#)).

## Build server application

Check this option to generate the server part of your application during the building phase. You must designate the location on your disk of the 4D Server application to be used. This 4D Server must correspond to the current platform (which will also be the platform of the server application).

### 4D Server location

Click on the [...] button and use the *Browse for folder* dialog box to locate the 4D Server application. In macOS, you must select the 4D Server package directly.

### Current version

Used to indicate the current version number for the application generated. You may then accept or reject connections by client applications according to their version number. The interval of compatibility for client and server applications is set using specific [XML keys](#)).

### Embed the project Users and Groups in built server application

**Preliminary Note:** The following terms are used in this section:

Name	Definition
Project directory file	<a href="#">directory.json</a> file located in the <a href="#">Settings folder</a> of the project
Application directory file	<a href="#">directory.json</a> file located in the <a href="#">Settings folder</a> of the built 4D Server
Data directory file	<a href="#">directory.json</a> file in the <a href="#">Data &gt; Settings folder</a>

When you check this option, the project directory file is copied to the application directory file at build time.

When you execute a built 4D Server application:

- If the server has a data directory file, it is loaded.
- If the server does not have a data directory file, the application directory file is loaded.

The application directory file is read-only. Modifications made to users, groups and permissions during server execution are stored in the data directory file. If no data directory file already exists, it is automatically created. If the application directory file was embedded, it is duplicated as data directory file.

Embedding the project directory file allows you to deploy a client/server application with a basic security user and group configuration. Subsequent modifications are added to the data directory file.

### Allow connection of Silicon Mac clients

When building a server on Windows, check this option to allow Apple Silicon clients to connect to your server application. You can then specify a path to the structure compiled for Apple Silicon/Intel.

To allow Apple Silicon clients to connect to a Server application built on Windows, you must first build a client application on macOS, with a project compiled for Apple Silicon

and Intel. This automatically creates a compiled structure, identical to the one created with the [Build compiled structure](#) option (without the related folders).

Then, you can copy that structure to your Windows machine, and use it to build the server application:

## Compiled structure location

Path to compiled structure of the Apple Silicon/Intel client application used to build a Windows Server (see [Allow connection of Silicon Mac clients](#)).

## Data linking mode

This option lets you choose the linking mode between the merged application and the local data file. Two data linking modes are available:

- **By application name** (default) - The 4D application automatically opens the most recently opened data file corresponding to the structure file. This allows you to move the application package freely on the disk. This option should generally be used for merged applications, unless you specifically need to duplicate your application.
- **By application path** - The merged 4D application will parse the application's *lastDataPath.xml* file and try to open the data file with an "executablePath" attribute that matches the application's full path. If such an entry is found, its corresponding data file (defined through its "dataFilePath" attribute) is opened. Otherwise, the last opened data file is opened (default mode).

For more information about the data linking mode, refer to the [Last data file opened](#) section.

## Build client application

Checking this option generates the client part of your application during the building phase.

You can check this option:

- along with the [Build server application](#) option to build matching server and client parts for the current platform and (optionally) include the automatic update archive files,
- without selecting the [Build server application](#) option, usually to build the update archive file to be selected from the "concurrent" platform when building the server part.

## 4D Volume Desktop Location

Designates the location on your disk of the 4D Volume Desktop application to be used to build the client part of your application.

The 4D Volume Desktop version number must match the 4D Developer Edition version number. For example, if you use 4D v19, you must select a 4D Volume Desktop v19.

The 4D Volume Desktop must correspond to the current platform (which will also be the platform of the client application). If you want to build a client application for the "concurrent" platform, you must carry out an additional build operation using a 4D

application running on that platform.

If you want the client application to connect to the server using a specific address (other than the server name published on the sub-network), you must use the `IPAddress` XML key in the buildapp.4DSettings file. For more information about this file, refer to the description of the [BUILD APPLICATION](#) command. You can also implement specific mechanisms in the event of a connection failure. The different scenarios proposed are described in the [Management of connections by client applications](#) paragraph.

## **Copy of client applications inside the server application**

The options of this area set up the mechanism for updating the client part(s) of your client/server applications using the network each time a new version of the application is generated. These options are only enabled when the **Build client application** option is checked.

- **Allow automatic update of Windows client application** - Check this option to build a `.4darchive` file that can be sent to your client applications on the Windows platform in case of update.
- **Allow automatic update of Macintosh client application** - Check this option to build a `.4darchive` file that can be sent to your client applications on the Macintosh platform in case of update.

The `.4darchive` is copied at the following location:

`<ApplicationName>_Build/Client Server executable/Upgrade4DClient/`

## **Selecting client archive for the concurrent platform**

You can check the **Allow automatic update...** option for client applications running on the concurrent platform. This option is only enabled if:

- the **Build server application** option is checked,
- the **Allow automatic update...** option for client applications running on the current platform is checked.

This feature requires that you click on the [...] button and designate the location on your disk of the file to use for the update. The file to select depends on the current server platform:

<b>Current server platform</b>	<b>Required file</b>	<b>Details</b>
macOS	Windows 4D Volume Desktop or Windows client update archive	By default, you select the <code>4D Volume Desktop</code> application for Windows. To select a <code>.4darchive</code> file previously built on Windows, press <b>Shift</b> while clicking on [...]
Windows	macOS client update archive	Select a signed <code>.4darchive</code> file previously built on macOS

You can build specific a `.4darchive` file on the concurrent platform by selecting only the [Build client application](#) and the appropriate [Allow automatic update...](#) option.

## **Displaying update notification**

The client application update notification is carried out automatically following the

server application update.

It works as follows: when a new version of the client/server application is built using the application builder, the new client portion is copied as a compressed file in the **Upgrade4DClient** subfolder of the **ApplicationName** Server folder (in macOS, these folders are included in the server package). If you have followed the process for generating a cross-platform client application, a *.4darchive* update file is available for each platform:

To trigger client application update notifications, simply replace the old version of the server application with the new one and then execute it. The rest of the process is automatic.

On the client side, when the “old” client application tries to connect to the updated server application, a dialog box is displayed on the client machine, indicating that a new version is available. The user can either update their version or cancel the dialog box.

- If the user clicks **OK**, the new version is downloaded to the client machine over the network. Once the download is complete, the old client application is closed and the new version is launched and connects to the server. The old version of the application is then placed in the machine’s recycle bin.
- If the user clicks **Cancel**, the update is cancelled; if the old version of the client application is not in the range of versions accepted by the server (please refer to the following paragraph), the application is closed and connection is impossible. Otherwise (by default), the connection is established.

## Forcing automatic updates

In some cases, you may want to prevent client applications from being able to cancel the update download. For example, if you used a new version of the 4D Server source application, the new version of the client application must absolutely be installed on each client machine.

To force the update, simply exclude the current version number of client applications (X-1 and earlier) in the version number range compatible with the server application. In this case, the update mechanism will not allow non-updated client applications to connect. For example, if the new version of the client-server application is 6, you can stipulate that any client application with a version number lower than 6 will not be allowed to connect.

The [current version number](#) is set on the Client/Server page of the Build Application dialog box. The intervals of authorized numbers are set in the application project using specific [XML keys](#).

## Update Error

If 4D cannot carry out the update of the client application, the client machine displays the following error message: "The update of the client application failed. The application is now going to quit."

There are many possible causes for this error. When you get this message, it is advisable to check the following parameters first off:

- **Pathnames** - Check the validity of the pathnames set in the application project via the Application builder dialog box or via XML keys (for example *ClientMacFolderToWin*). More particularly, check the pathnames to the versions of 4D Volume Desktop.

- **Read/write privileges** - On the client machine, check that the current user has write access rights for the client application update.

## Generated files

Once a client/server application is built, you will find a new folder in the destination folder named **Client Server executable**. This folder contains two subfolders, `<ApplicationName>Client` and `<ApplicationName>Server`.

These folders are not generated if an error occurs. In this case, open the [log file](#) in order to find out the cause of the error.

The `<ApplicationName>Client` folder contains the client portion of the application corresponding to the execution platform of the application builder. This folder must be installed on each client machine. The `<ApplicationName>Server` folder contains the server portion of the application.

The contents of these folders vary depending on the current platform:

- *Windows* - Each folder contains the application executable file, named `<ApplicationName>Client.exe` for the client part and `<ApplicationName>Server.exe` for the server part as well as the corresponding `.rsr` files. The folders also contain various files and folders necessary for the applications to work and customized items that may be in the original 4D Volume Desktop and 4D Server folders.
- *macOS* - Each folder contains only the application package, named `<ApplicationName> Client` for the client part and `<ApplicationName> Server` for the server part. Each package contains all the necessary items for the application to work. Under macOS, launch a package by double-clicking it.

The macOS packages built contain the same items as the Windows subfolders. You can display their contents (**Control+click** on the icon) in order to be able to modify them.

If you checked the “Allow automatic update of client application” option, an additional subfolder called *Upgrade4DClient* is added in the `<ApplicationName>Server` folder/package. This subfolder contains the client application in macOS and/or Windows format as a compressed file. This file is used during the automatic client application update.

## Location of Web files

If the server and/or client part of your double-clickable application is used as a Web server, the files and folders required by the server must be installed in specific locations. These items are the following:

- *cert.pem* and *key.pem* files (optional): These files are used for TLS connections and by data encryption commands,
- Default Web root folder (WebFolder).

Items must be installed:

- **on Windows**

- **Server application** - in the `Client Server executable/<ApplicationName>Server/Server Database` subfolder.
- **Client application** - in the `Client Server`

`executable/<ApplicationName>Client` subfolder.

- **on macOS**

- **Server application** - next to the `<ApplicationName>Server` software package.
- **Client application** - next to the `<ApplicationName>Client` software package.

## Embedding a single-user client application

4D allows you to embed a compiled structure in the Client application. This feature can be used, for example, to provide users with a "portal" application, that gives access to different server applications thanks to the `OPEN DATABASE` command executing a `.4dlink` file.

To enable this feature, add the `DatabaseToEmbedInClientWinFolder` and/or `DatabaseToEmbedInClientMacFolder` keys in the `buildApp` settings file. When one of these keys is present, the client application building process generates a single-user application: the compiled structure, instead of the `EnginedServer.4Dlink` file, is placed in the "Database" folder.

- If a default data folder exists in the single-user application, a licence is embedded.
- If no default data folder exists in the single-user application, it will be executed without data file and without licence.

The basic scenario is:

1. In the Build application dialog box, select the "Build compiled structure" option to produce a `.4DZ` or `.4DC` for the application to be used in single-user mode.
2. In the `buildApp.4DSettings` file of the client-server application, use following xml key(s) to indicate the path to the folder containing the compiled single user application:
  - `DatabaseToEmbedInClientWinFolder`
  - `DatabaseToEmbedInClientMacFolder`
3. Build the client-server application. This will have following effects:
  - the whole folder of the single user application is copied inside the "Database" folder of the merged client
  - the `EnginedServer.4Dlink` file of the "Database" folder is not generated
  - the `.4DC`, `.4DZ`, `.4DIndy` files of the single user application copy are renamed using the name of the merged client
  - the `PublishName` key is not copied in the `info.plist` of the merged client
  - if the single-user application does not have a "Default data" folder, the merged client will run with no data.

Automatic update 4D Server features ([Current version](#) number, `SET UPDATE FOLDER` command...) work with single-user application as with standard remote application. At connection, the single-user application compares its `CurrentVers` key to the 4D Server version range. If outside the range, the updated client application is downloaded from the server and the Updater launches the local update process.

## Customizing client and/or server cache folder names

Client and server cache folders are used to store shared elements such as resources or

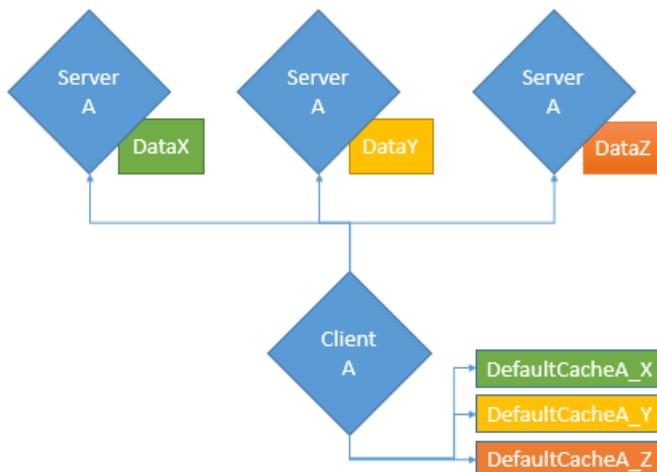
components. They are required to manage exchanges between server and remote clients. Client/server applications use default pathnames for both client and server system cache folders.

In some specific cases, you might need to customize the names of these folders to implement specific architectures (see below). 4D provides you with the `ClientServerSystemFolderName` and `ServerStructureFolderName` keys to be set in the `buildApp` settings file.

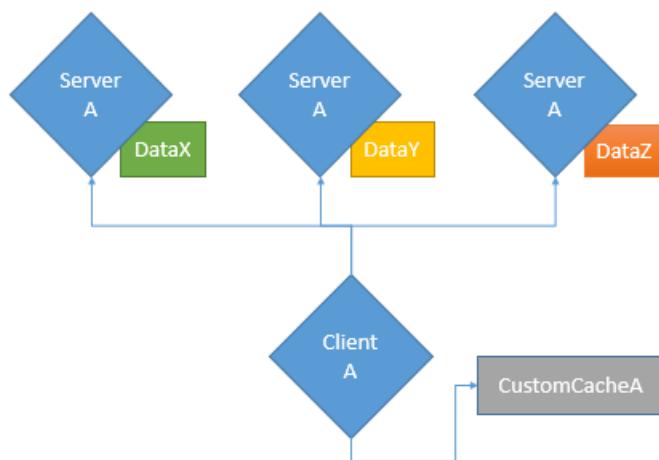
## Client cache folder

Customizing the client-side cache folder name can be useful when your client application is used to connect to several merged servers which are similar but use different data sets. In this case, to save multiple unnecessary downloads of identical local resources, you can use the same custom local cache folder.

- Default configuration (for each connection to a server, a specific cache folder is downloaded/updated):



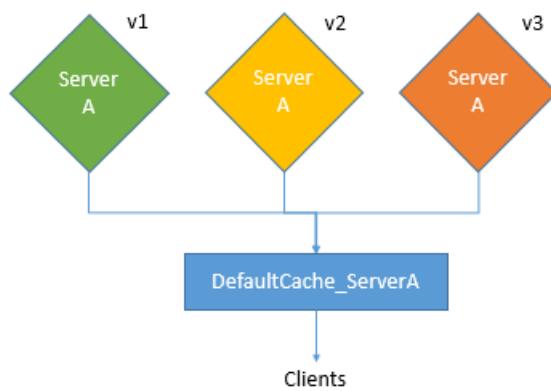
- Using the `ClientServerSystemFolderName` key (a single cache folder is used for all servers):



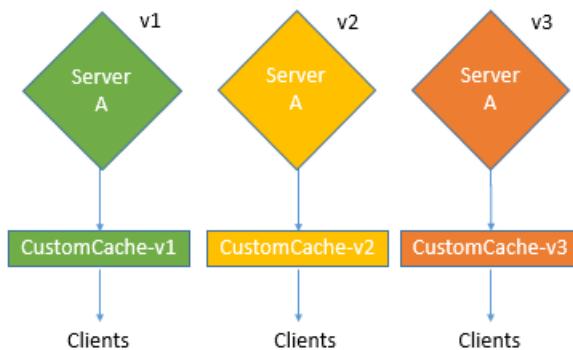
## Server cache folder

Customizing the server-side cache folder name is useful when you run several identical server applications built with different 4D versions on the same computer. If you want each server to use its own set of resources, you need to customize the server cache folder.

- Default configuration (*same server applications share the same cache folder*):



- Using the `ServerStructureFolderName` key (*a dedicated cache folder is used for each server application*):



## Plugins & components page

On this tab, you set each **plug-in**, **component**, and **module** that you will use in your stand-alone or client/server application.

The page lists the elements loaded by the current 4D application:

- **Active** column - Indicates that the items will be integrated into the application package built. All the items are checked by default. To exclude a plug-in, a component, or a module, deselect the check box next to it.
- **Plugins and components** column - Displays the name of the plug-in/component/module.
- **ID** column - Displays the element's identification number (if any).
- **Type** column - Indicates the type of item: Plug-in, Component, or Module.

### Adding plug-ins or components

If you want to integrate other plug-ins or components into the executable application, you just need to place them in a **PlugIns** or **Components** folder next to the 4D Volume Desktop application or next to the 4D Server application. The mechanism for copying the contents of the source application folder (see [Customizing the 4D Volume Desktop folder](#)) can be used to integrate any type of file into the executable application.

If there is a conflict between two different versions of the same plug-in (one loaded by 4D and the other located in the source application folder), priority goes to the plug-in

installed in the 4D Volume Desktop/4D Server folder. However, if there are two instances of the same component, the application will not open.

The use of plug-ins and/or components in a deployment version may require license numbers.

## Deselecting modules

A module is a built-in code library used by 4D to control specific features. If you know that your built application does not use any of the features covered by a module, you can deselect it in the list to reduce the size of your application files.

**Warning:** Deselecting a module could prevent your built application from working as expected. If you are not 100% certain that a module is never called by your application, it is recommended to keep it selected.

The following optional modules can be deselected:

- **CEF:** Chromium embedded library. It is necessary to run [Web areas](#) that use the embedded rendering engine and [4D View Pro areas](#). Calling such areas when CEF is deselected will display blank areas and/or generate errors.
- **MeCab:** Library used for text indexing in Japanese language (see this [settings paragraph](#)). Deselecting this module will force text indexes to be rebuilt in Japanese language.

If you deselect MeCab for an application in Japanese language used on heterogeneous platforms, make sure to deselect it on both client/server build and [client application build](#) (for the concurrent platform), otherwise major malfunctions will occur in the application.

- **PHP:** Necessary to use PHP features and commands in 4D (see this [settings paragraph](#)).
- **SpellChecker:** Used for built-in [spellchecking features](#) and commands available for input areas and 4D Write Pro areas.
- **4D Updater:** Controls the [automatic update](#) of client parts and is used by the `SET UPDATE FOLDER` command for [automated server updates](#).

## Licenses & Certificate page

The Licences & Certificate page can be used to:

- designate the license number(s) that you want to integrate into your single-user stand-alone application
- sign the application by means of a certificate in macOS.

### Licenses

This tab displays the list of available deployment licenses that you can integrate into your application. By default, the list is empty. You must explicitly add your *4D Developer Professional* license as well as each *4D Desktop Volume* license to be used in the application built. You can add another 4D Developer Professional number and its associated licenses other than the one currently being used.

To remove or add a license, use the **[+]** and **[ - ]** buttons at the bottom of the window.

When you click on the **[+]** button, an open file dialog box appears displaying by default the contents of the *Licenses* folder of your machine. For more information

about the location of this folder, refer to the [Get 4D folder](#) command.

You must designate the files that contain your Developer license as well as those containing your deployment licenses. These files were generated or updated when the *4D Developer Professional* license and the *4D Desktop Volume* licenses were purchased.

Once you have selected a file, the list will indicate the characteristics of the license that it contains.

- **License #** - Product license number
- **License** - Name of the product
- **Expiration date** - Expiration date of the license (if any)
- **Path** - Location on disk

If a license is not valid, a message will warn you.

You can designate as many valid files as you want. When building an executable application, 4D will use the most appropriate license available.

Dedicated "R" licenses are required to build applications based upon "R-release" versions (license numbers for "R" products start with "R-4DDP").

After the application is built, a new deployment license file is automatically included in the Licenses folder next to the executable application (Windows) or in the package (macOS).

## OS X signing certificate

The application builder can sign merged 4D applications under macOS (single-user applications, components, 4D Server and client parts under macOS). Signing an application authorizes it to be executed using the Gatekeeper functionality of macOS when the "Mac App Store and identified Developers" option is selected (see "About Gatekeeper" below).

- Check the **Sign application** option to include certification in the application builder procedure for OS X. 4D will check the availability of elements required for certification when the build occurs:

The screenshot shows a dialog box titled "OS X signing certificate". It contains a checked checkbox labeled "Sign application". Below the checkbox is a note: "By default, Apple requires that applications downloaded from the internet be signed by the developer." At the bottom, there is a text input field labeled "Name of certificate:" followed by a "Generate self-signed certificate" button.

This option is displayed under both Windows and macOS, but it is only taken into account for macOS versions.

- **Name of certificate** - Enter the name of your developer certificate validated by Apple in this entry area. The certificate name is usually the name of the certificate in the Keychain Access utility (part in red in the following example):

To obtain a developer certificate from Apple, Inc., you can use the commands of the Keychain Access menu or go here:

<http://developer.apple.com/library/mac/#documentation/Security/Conceptual/CodeSigningGuide/Procedures/Procedures.html>.

This certificate requires the presence of the Apple codesign utility, which is provided by default and usually located in the "/usr/bin/" folder. If an error occurs, make sure that this utility is present on your disk.

- **Generate self-signed certificate** - runs the "Certificate Assistant" that allows you to generate a self-signed certificate. If you do not have an Apple developer certificate, you need to provide a self-signed certificate. With this certificate, no alert message is displayed if the application is deployed internally. If the application is deployed externally (i.e. through http or email), at launch macOS displays an alert message that the application's developer is unidentified. The user can "force" the opening of the application. In the "Certificate Assistant", be sure to select the appropriate options:

4D recommends to subscribe to the Apple Developer Program to get access to Developer Certificates that are necessary to notarize applications (see below).

## About Gatekeeper

Gatekeeper is a security feature of OS X that controls the execution of applications downloaded from the Internet. If a downloaded application does not come from the Apple Store or is not signed, it is rejected and cannot be launched.

On Apple Silicon machines, 4D [components](#) need to be actually signed. An unsigned component will generate an error at application startup ("lib4d-arm64.dylib can't be opened...").

The **Sign application** option of the 4D application builder lets you generate applications and components that are compatible with this option by default.

## About Notarization

Application notarization is highly recommended by Apple as of macOS 10.14.5 (Mojave) and 10.15 (Catalina), since non-notarized applications deployed via the internet are blocked by default.

The 4D [built-in signing features](#) have been adapted to meet all of Apple's requirements to allow using the Apple notary service. The notarization itself must be conducted by the developer and is independent from 4D (note also that it requires installing Xcode). Please refer to [this 4D blog post](#) that provides a step-by-step description of the notarization process.

For more information on the notarization concept, please refer to [this page on the Apple developer website](#).

## Customizing application icons

4D associates a default icon with stand-alone, server, and client applications, however you can customize the icon for each application.

- **macOs** - When building a double-clickable application, 4D handles the customizing of the icon. In order to do this, you must create an icon file (icns type), prior to building the application file, and place it next to the project folder.

Apple, Inc. provides a specific tool for building *icns* icon files (for more information, please refer to [Apple documentation](#)).

Your icon file must have the same name as the project file and include the *.icns* extension. 4D automatically takes this file into account when building the double-clickable application (the *.icns* file is renamed *ApplicationName.icns* and copied into the Resources folder; the *CFBundleFileIcon* entry of the *info.plist* file is updated).

- **Windows** - When building a double-clickable application, 4D handles the customizing of its icon. In order to do this, you must create an icon file (*.ico* extension), prior to building the application file, and place it next to the project folder.

Your icon file must have the same name as the project file and include the *.ico* extension. 4D automatically takes this file into account when building the double-clickable application.

You can also set specific [XML keys](#) in the *buildApp.4DSettings* file to designate each icon to use. The following keys are available:

- RuntimeVLIconWinPath
- RuntimeVLIconMacPath
- ServerIconWinPath
- ServerIconMacPath
- ClientMacIconForMacPath
- ClientWinIconForMacPath
- ClientMacIconForWinPath
- ClientWinIconForWinPath

## Management of data file(s)

### Opening the data file

When a user launches a merged application or an update (single-user or client/server applications), 4D tries to select a valid data file. Several locations are examined by the application successively.

The opening sequence for launching a merged application is:

1. 4D tries to open the last data file opened, [as described below](#) (not applicable during initial launch).
2. If not found, 4D tries to open the data file in a default data folder next to the *.4DZ* file in read-only mode.
3. If not found, 4D tries to open the standard default data file (same name and same location as the *.4DZ* file).
4. If not found, 4D displays a standard "Open data file" dialog box.

### Last data file opened

#### Path of last data file

Any standalone or server applications built with 4D stores the path of the last data file opened in the application's user preferences folder.

The location of the application's user preferences folder corresponds to the path returned by the following statement:

```
userPrefs:=Get 4D folder(Active 4D Folder)
```

The data file path is stored in a dedicated file, named *lastDataPath.xml*.

Thanks to this architecture, when you provide an update of your application, the local user data file (last data file used) is opened automatically at first launch.

This mechanism is usually suitable for standard deployments. However, for specific needs, for example if you duplicate your merged applications, you might want to change the way that the data file is linked to the application (described below).

## Configuring the data linking mode

With your compiled applications, 4D automatically uses the last data file opened. By default, the path of the data file is stored in the application's user preferences folder and is linked to the **application name**.

This may be unsuitable if you want to duplicate a merged application intended to use different data files. Duplicated applications actually share the application's user preferences folder and thus, always use the same data file -- even if the data file is renamed, because the last file used for the application is opened.

4D therefore lets you link the data file path to the application path. In this case, the data file will be linked using a specific path and will not just be the last file opened. You therefore link your data **by application path**.

This mode allows you to duplicate your merged applications without breaking the link to the data file. However, with this option, if the application package is moved on the disk, the user will be prompted for a data file, since the application path will no longer match the "executablePath" attribute (after a user has selected a data file, the *lastDataPath.xml* file is updated accordingly).

*Duplication when data linked by application name:*

*Duplication when data linked by application path:*

You can select the data linking mode during the build application process. You can either:

- Use the [Application page](#) or [Client/Server page](#) of the Build Application dialog box.
- Use the **LastDataPathLookup** XML key (single-user application or server application).

## Defining a default data folder

4D allows you to define a default data file at the application building stage. When the application is launched for the first time, if no local data file is found (see [opening sequence described above](#)), the default data file is automatically opened silently in read-only mode by 4D. This gives you better control over data file creation and/or opening when launching a merged application for the first time.

More specifically, the following cases are covered:

- Avoiding the display of the 4D "Open Data File" dialog box when launching a new or updated merged application. You can detect, for example at startup, that the default data file has been opened and thus execute your own code and/or dialogs to create or select a local data file.
- Allowing the distribution of merged applications with read-only data (for demo applications, for instance).

To define and use a default data file:

- You provide a default data file (named "Default.4DD") and store it in a default

folder (named "Default Data") inside the application project folder. This file must be provided along with all other necessary files, depending on the project configuration: index (.4DIndx), external Blobs, journal, etc. It is your responsibility to provide a valid default data file. Note however that since a default data file is opened in read-only mode, it is recommended to uncheck the "Use Log File" option in the original structure file before creating the data file.

- When the application is built, the default data folder is integrated into the merged application. All files within this default data folder are also embedded.

The following graphic illustrates this feature:

When the default data file is detected at first launch, it is silently opened in read-only mode, thus allowing you to execute any custom operations that do not modify the data file itself.

## Management of client connection(s)

The management of connections by client applications covers the mechanisms by which a merged client application connects to the target server, once it is in its production environment.

### Connection scenario

The connection procedure for merged client applications supports cases where the dedicated server is not available. The startup scenario for a 4D client application is the following:

1. If valid connection information is stored in the "EnginedServer.4DLink" file within the client application, the client application connects to the specified server address.  
OR  
The client application tries to connect to the server using the discovery service (based upon the server name, broadcasted on the same subnet).
2. If this fails, the client application tries to connect to the server using information stored in the application's user preferences folder ("lastServer.xml" file, see last step).
3. If this fails, the client application displays a connection error dialog box.
  - If the user clicks on the **Select...** button (when allowed by the 4D developer at the build step, see below), the standard "Server connection" dialog box is displayed.
  - If the user clicks on the **Quit** button, the client application quits.
4. If the connection is successful, the client application saves this connection information in the application's user preferences folder for future use.

The whole procedure is described in the following diagram:

### Storing the last server path

The last used and validated server path is automatically saved in a file named "lastServer.xml" in the application's user preferences folder. This folder is stored at the following location:

```
userPrefs:=Get 4D folder (Active 4D Folder)
```

This mechanism addresses the case where the primary targeted server is temporary unavailable for some reason (maintenance mode for example). When this case occurs for the first time, the server selection dialog box is displayed (if allowed, see below) and the user can manually select an alternate server, whose path is then saved if the connection is successful. Any subsequent unavailability would be handled automatically through the "lastServer.xml" path information.

- When client applications cannot permanently benefit from the discovery service, for example because of the network configuration, it is recommended that the developer provide a host name at build time using the [IPAddress](#) key in the "BuildApp.4DSettings" file. The mechanism addresses cases of temporary unavailability.
- Pressing the **Alt/Option** key at startup to display the server selection dialog box is still supported in all cases.

### Availability of the server selection dialog box in case of error

You can choose whether or not to display the standard server selection dialog box on merged client applications when the server cannot be reached. The configuration depends on the value of the [ServerSelectionAllowed](#) XML key on the machine where the application was built:

- **Display of an error message with no access possible to the server selection dialog box.** Default operation. The application can only quit.  
`ServerSelectionAllowed: False` or key omitted
- **Display of an error message with access to the server selection dialog box possible.** The user can access the server selection window by clicking on the **Select...** button. `ServerSelectionAllowed: True`

## Automatic updating of server or single-user applications

In principle, updating server applications or merged single-user applications require user intervention (or programming custom system routines): whenever a new version of the merged application is available, you have to exit the application in production and manually replace the old files with the new ones; then restart the application and select the current data file.

You can automate this procedure to a large extent using the following language commands: [SET\\_UPDATE\\_FOLDER](#), [RESTART\\_4D](#), and also [Get last update log path](#) for monitoring operations. The idea is to implement a function in your 4D application triggering the automatic update sequence described below. It can be a menu command or a process running in the background and checking at regular intervals for the presence of an archive on a server.

You also have XML keys to elevate installation privileges so that you can use protected files under Windows (see the [4D XML Keys BuildApplication](#) manual).

Here is the scenario for updating a server or merged single-user application:

1. You transfer, for example using an HTTP server, the new version of the server application or the merged single-user application onto the machine in production.

2. In the application in production, you call the `SET UPDATE FOLDER` command: this command designates the location of the folder where the "pending" update of the current application is found. Optionally, you can copy in this folder the custom elements of the version in production (user files).
3. In the application in production, call the `RESTART 4D` command: this command automatically triggers execution of a utility program named "updater" that exits the current application, replaces it using the "pending" update if one is specified, and restarts the application with the current data file. The former version is renamed.

This sequence is compatible with Windows server applications run as a Service.

## Update log

The installation procedure produces a log file detailing the update operations of merged applications (client, server or single-user) on the target machines. This file is useful for analyzing any errors that occur during the installation process.

The update log is named `YYYY-MM-DD_HH-MM-SS_log_X.txt`, for example, `2021-08-25_14-23-00_log_1.txt` for a file created on August 25, 2021 at 14:23.

This file is created in the "Updater" application folder, within the system user folder. You can find out the location of this file at any time using the [Get last update log path](#) command.