ConfigurationDesk

# Automating Tool Handling

For ConfigurationDesk 6.7

Release 2021-A – May 2021

dSPACE

## How to Contact dSPACE


| | |
|---|---|
| Mail: | dSPACE GmbH<br>Rathenaustraße 26<br>33102 Paderborn<br>Germany |
| Tel.: | +49 5251 1638-0 |
| Fax: | +49 5251 16198-0 |
| E-mail: | info@dspace.de |
| Web: | http://www.dspace.com |


## How to Contact dSPACE Support

If you encounter a problem when using dSPACE products, contact your local dSPACE representative:

- Local dSPACE companies and distributors: http://www.dspace.com/go/locations
- For countries not listed, contact dSPACE GmbH in Paderborn, Germany.
  Tel.: +49 5251 1638-941 or e-mail: support@dspace.de

You can also use the support request form: http://www.dspace.com/go/supportrequest. If you are logged on to mydSPACE, you are automatically identified and do not need to add your contact details manually.

If possible, always provide the relevant dSPACE License ID or the serial number of the CmContainer in your support request.

## Software Updates and Patches

dSPACE strongly recommends that you download and install the most recent patches for your current dSPACE installation. Visit http://www.dspace.com/go/patches for software updates and patches.

## Important Notice

# Contents

# About This Document

**Content**

This document gives you detailed information on ConfigurationDesk's automation interface.

**Note**

The PDF version of this document does not contain graphical representations of the API elements in the chapter ConfigurationDesk API Reference on page 127. For graphical representations, refer to dSPACE Help.

**Required knowledge**

You should be familiar with performing the tasks to be automated in ConfigurationDesk without automation.

Knowledge in handling the host PC and the Microsoft Windows operating system is presupposed. You should also be familiar with a programming language such as Python, C, or C#.

**Symbols**

dSPACE user documentation uses the following symbols:

| Symbol | Description |
|---|---|
| DANGER | Indicates a hazardous situation that, if not avoided, will result in death or serious injury. |
| WARNING | Indicates a hazardous situation that, if not avoided, could result in death or serious injury. |
| CAUTION | Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury. |
| NOTICE | Indicates a hazard that, if not avoided, could result in property damage. |
| Note | Indicates important information that you should take into account to avoid malfunctions. |

| Symbol | Description |
|---|---|
| Tip | Indicates tips that can make your work easier. |
|  | Indicates a link that refers to a definition in the glossary, which you can find at the end of the document unless stated otherwise. |
|  | Precedes the document title in a link that refers to another document. |

**Naming conventions**

dSPACE user documentation uses the following naming conventions:

**%name%** Names enclosed in percent signs refer to environment variables for file and path names.

**< >** Angle brackets contain wildcard characters or placeholders for variable file and path names, etc.

**Special folders**

Some software products use the following special folders:

**Common Program Data folder** A standard folder for application-specific configuration data that is used by all users.

`%PROGRAMDATA%\dSPACE\<InstallationGUID>\<ProductName>`

or

`%PROGRAMDATA%\dSPACE\<ProductName>\<VersionNumber>`

**Documents folder** A standard folder for user-specific documents.

`%USERPROFILE%\Documents\dSPACE\<ProductName>\<VersionNumber>`

**Local Program Data folder** A standard folder for application-specific configuration data that is used by the current, non-roaming user.

`%USERPROFILE%\AppData\Local\dSPACE\<InstallationGUID>\<ProductName>`

**Accessing dSPACE Help and PDF Files**

After you install and decrypt dSPACE software, the documentation for the installed products is available in dSPACE Help and as PDF files.

**dSPACE Help (local)** You can open your local installation of dSPACE Help:

- On its home page via Windows Start Menu
- On specific content using context-sensitive help via **F1**

**dSPACE Help (Web)** You can access the Web version of dSPACE Help at www.dspace.com/go/help.

To access the Web version, you must have a *mydSPACE* account.

**PDF files** You can access PDF files via the  icon in dSPACE Help. The PDF opens on the first page.

# Automating ConfigurationDesk

**Where to go from here**

Information in this section

# Introduction to Automating ConfigurationDesk

**Objective**

ConfigurationDesk lets you automate most of its features via its automation interface.

**Tip**

If you are not an experienced Python user and require basic information on using Python scripts, refer to Basics on Python Relevant for Automation on page 95.

**Python demo files**

ConfigurationDesk comes with a number of Python demo scripts, which you can find in the `ToolAutomation` subfolder of the documents folder after ConfigurationDesk was started for the first time. You can run these scripts in ConfigurationDesk's **Internal Interpreter** or an external interpreter, such as Python.exe.

**Tip**

For help on translating Python examples into C# or VB, refer to Examples of Translating Python Code into Different Programming Languages on page 103.

## Overview of the Automation Interface

**Features of ConfigurationDesk's automation interface**

ConfigurationDesk lets you automate most of its features:

- Creating a new project and application
- Adding and configuring a platform/device
- Accessing and editing data sets
- Customizing the user interface
- Configuring model port block data (refer to General Information on the Model Interface Blockset (Model Interface Package for Simulink - Modeling Guide 🕮))
- Setting the model save mode (refer to Handling Model Interface Package for Simulink Preferences (Model Interface Package for Simulink - Modeling Guide 🕮))

**Interface API type**

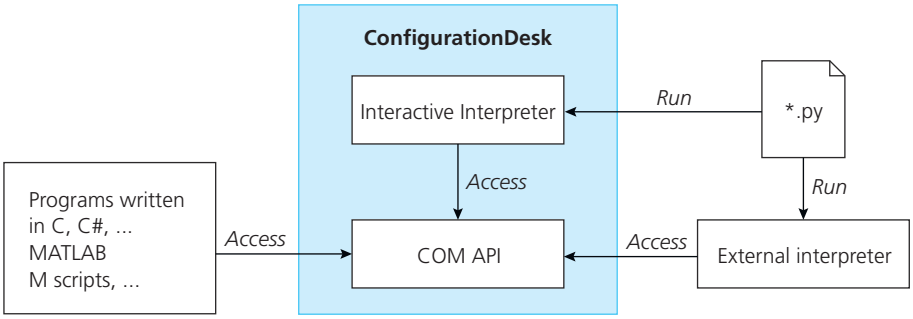
ConfigurationDesk's automation interface is an API which is implemented as a COM object model. The Microsoft Component Object Model (COM) supports

communication between objects from different applications. It can be used by any COM-compatible application, regardless of the programming language in which it was developed.

For more information on the elements and structure of the object model refer to Basics on the Object Model in ConfigurationDesk on page 36.

## Accessing ConfigurationDesk's automation interface

The illustration below displays different ways of accessing the COM API provided by ConfigurationDesk.




You can access ConfigurationDesk's automation interface:

- Via ConfigurationDesk's **Internal Interpreter**
- Via an external interpreter
- Via applications or further scripting environments

## Supported automation languages

ConfigurationDesk supports automation by any COM-compatible application.

You can automate ConfigurationDesk, for example, by using the following scripting environments:

- Python scripts running in ConfigurationDesk's **Internal Interpreter**.
- Python scripts running in an external interpreter, for example, Python.exe.
- M scripts running in MATLAB.
- VBA scripts running in Microsoft Office.

You can automate ConfigurationDesk by applications developed, for example, with the following programming languages:

- C#
- C++
- Visual Basic

**Tip**

If you use a programming environment that supports the Intellisense feature such as Microsoft Visual Studio, this feature provides information about ConfigurationDesk's API. Intellisense displays the properties and methods of each interface together with the description that is also available via the ConfigurationDesk API Reference on page 127.

# Ways to use Python Scripts in ConfigurationDesk

**Objective**

ConfigurationDesk offers different ways to write, edit, and run Python scripts:

- You can write Python scripts and run them via ConfigurationDesk's **Internal Interpreter**.
- You can write Python scripts and execute them via an external interpreter.
- You can write and edit Python scripts in the Source Code Editor.

**Where to go from here**

Information in this section



# Using the Interpreter

**Introduction**

The **Interpreter** lets you edit and execute Python commands and run Python scripts.

**Where to go from here**

Information in this section

# Basics on the Interpreter

**Introduction**

The Interpreter lets you edit and execute Python commands and run Python scripts.

**Interpreter user interface**

You can access ConfigurationDesk's automation interface by entering commands interactively or running scripts in the Interpreter.


**Editing features**

You can use the Interpreter to edit and run Python commands. ConfigurationDesk provides a set of features that makes editing easier and more efficient.

**Syntax highlighting** Helps you distinguish between the Python syntax items in your commands by highlighting them. For instructions, refer to How to Specify Syntax Highlighting on page 18.

**Auto completion** Completes Python variables, functions, and object attributes automatically so that you can save time and avoid spelling mistakes. For instructions, refer to How to Enable Auto Completion on page 20 and How to Use Auto Completion on page 21.

**Auto indentation** In Python, multi-line commands are introduced by a colon and their scopes are declared by indentation. The indentation indicates the structure of a multi-line command. The Interpreter indents the next line automatically. The indent depth of the line depends on the number of control structures or command blocks. You can decrease the indent depth by entering an empty line or pressing the **Backspace** key: for example, to close an `If` branch.

**Command history** The Interpreter stores a command history to let you quickly execute already executed commands during the current work session.

During a work session, you might often repeat some commands with only minor changes. All the commands entered at the command prompt are stored in a command history. You can navigate through the command history by using the shortcut key combinations **Ctrl+Up/Down** to move up or down in the command history, and **Ctrl+Home/End** to go to the first or last command in the command history.

**Shortcut keys** The Interpreter supports the use of shortcut keys for Interpreter commands. For a list of the supported shortcut keys, refer to Interpreter (ConfigurationDesk User Interface Reference 🕮).

**Find/Copy/Cut/Paste** You can use the standard Windows commands Find, Copy, Cut and Paste from the context menu of the Interpreter window or via shortcut keys. This feature helps you edit your commands more efficiently.

**Drag & Drop** You can move or copy selected text easily via drag & drop in the Interpreter controlbar.

The Interpreter controlbar provides two kinds of drag & drop methods according to the position of the text:

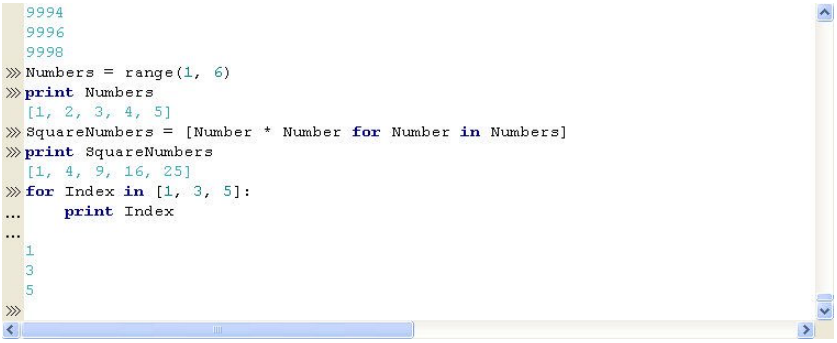
- Input area

  Text in the input area is the text in the current input lines. You can select text in the input area and move or copy it via drag & drop. You can also copy text from other applications to the input area via drag & drop.

- History area

  Text in the history area is the text before the current input lines. You can select text from the history area and copy it to your current input line by pressing the *Ctrl* key and using drag & drop at the same time.

The following illustration shows an output example:



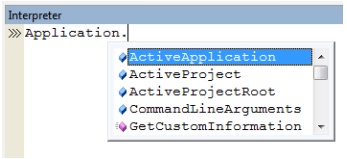

The lowest line is the current input line. The lines above are history lines. The following symbols are used:

| Symbol | Description |
|---|---|
| >>> | Input line.<br>The lowermost input line is the current input line. The input lines above are history input lines. |
| ... | Input line continuation.<br>If you enter, for example, a control loop such as `for` and finish the line with a colon, the Interpreter automatically adds a new indented line. |

| Symbol | Description |
|---|---|
| | To add further lines enter code and then press **Enter**.<br>To execute the multi line command leave the line empty and press **Enter**. |
| | Output line. |

**Application object**

The ConfigurationDesk API provides the **Application** interface. This interface gives you direct access to the running ConfigurationDesk application.




**Tip**

You can use the Application interface as starting point to browse the ConfigurationDesk API Reference. Refer to ICaApplicationMain <<Interface>> on page 192.

**Running scripts**

You can run Python scripts directly in the **Interpreter**. For instructions, refer to How to Run Scripts on page 23.

**Importing scripts**

You can use external variables and methods in your commands and Python scripts by importing scripts. The variables and methods defined in the scripts are loaded into the **Interpreter**'s namespace. For instructions, refer to How to Import a Python Module to the Interpreter Namespace on page 22.

**Specifying the Python path**

When you import a Python module, the **Interpreter** searches for it in the folders of the Python path. You can list the folders and specify the Python path. For instructions, refer to How to Specify the Python Path on page 24.

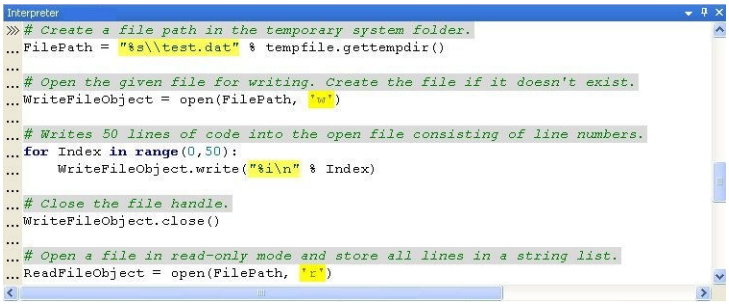
# How to Specify Syntax Highlighting

**Objective**

You can specify syntax highlighting to distinguish between Python syntax items in your commands.

**Basics**

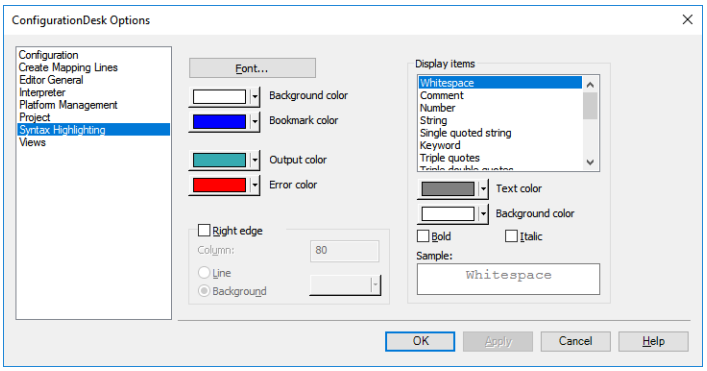
The Interpreter lets you specify general font and color settings, such as the interpreter's background color or the text color for errors. You can also specify how to display specific syntax items in the Python code. The syntax highlighting feature makes it easy to distinguish between the syntax items in the command lines.




**Method**

**To specify syntax highlighting for a display item**

1 On the File ribbon, click Options and change to the Syntax Highlighting page.




2 Specify general settings, such as the font or background color, to be used in the Interpreter.

3 From the Display Items list, select an item such as Comment or String, and configure the item's color and font settings.

4 Specify if and how a right edge is visualized in the Interpreter. The Interpreter can display a line or a background color to visualize that a code line exceeds a specified column limit.

5 Click OK to close the dialog.

**Result**

Syntax highlighting of a display item is specified.

**Related topics**

Basics

Basics on the Interpreter ..... 16

References

Syntax Highlighting Page (ConfigurationDesk User Interface Reference 🕮)

# How to Enable Auto Completion

**Objective**

To complete commands automatically in ConfigurationDesk's Interpreter window, you have to enable the auto completion feature.

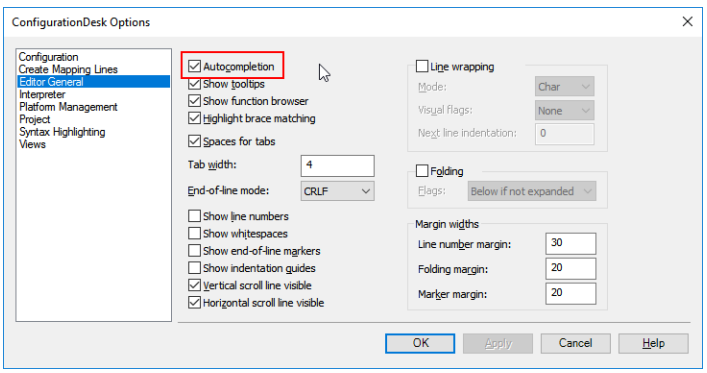
**Basics**

ConfigurationDesk's Interpreter includes an auto completion feature that makes typing commands easy. This feature helps you enter the names of variables, methods, and objects correctly and quickly. You do not have to remember the full name or worry about the spelling mistakes.

**Method**

**To enable/disable auto completion**

1. On the File ribbon, click Options and change to the Editor General page.
2. On the Editor General page, select the Autocompletion checkbox to enable autocompletion or clear the checkbox to disable it.




3. Click OK.

**Result**

You have enabled/disabled auto completion in the Interpreter.

**Related topics**

Basics

Basics on the Interpreter....16

HowTos

How to Use Auto Completion....21

# How to Use Auto Completion

**Objective**

To edit commands in the **Interpreter** easily and more efficiently, you can use the auto completion feature in ConfigurationDesk.

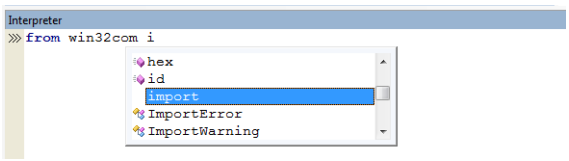
**Method**

**To use auto completion**

1. In the **Interpreter** controlbar, enter the beginning letters of a Python variable/method/object of any length.
2. Press **Ctrl** + **SPACE**.

   The **Interpreter** completes the matching Python variable/method/object name automatically.

   If the result is ambiguous, a drop-down list appears with the Python variables/methods/objects that are currently known to the Interpreter's namespace.




3. To complete the command, select the entry and press **Return**.

   **Tip**

   You can also use the **Tab** key to select the entry and press **Return** or you can double-click the entry.

   To close the selection list, press **Esc**.

**Result**

You have completed the Python variable, method, or object automatically.

**Related topics**

Basics

Basics on the Interpreter ..... 16

HowTos

How to Enable Auto Completion ..... 20

# How to Import a Python Module to the Interpreter Namespace

**Objective**

You can import a Python module to use its variables and methods in the current command and script.

**Basics**

You can use external variables and methods directly in your current commands and Python scripts. To do so, you have to import Python modules defining variables and methods.

**Note**

Unlike the `import <module_name>` Python command, the Import Module command overwrites the module if it was imported before, i.e., you do not need to clear the Interpreter namespace to reload a module. For details about namespace, refer to Clear Namespace (ConfigurationDesk User Interface Reference 🕮).

**Method**

**To import a Python module to the Interpreter namespace**

1 On the Automation ribbon, click Interpreter – Import Module to open the Import Module dialog.

2 In the Import Module dialog, specify the Python file.


3 Click OK.

**Result**

The module is imported into the Interpreter's namespace.

## How to Run Scripts

**Objective**

You can run a Python script in ConfigurationDesk's Interpreter to execute Python commands automatically to do tasks such as building a project or validating specified elements.

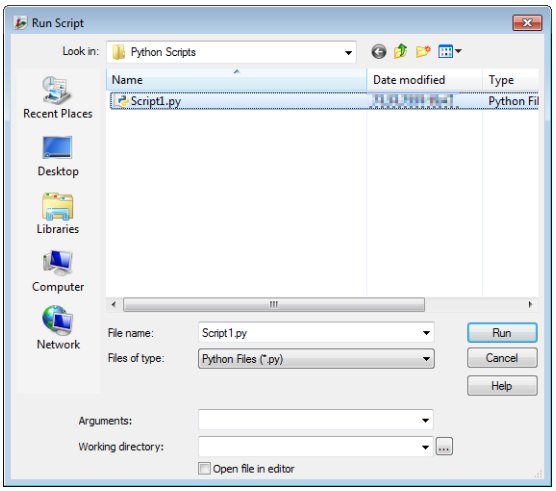
**Method**

**To run a Python script**

1 From the context menu of the Interpreter, select Run Script, or press **Ctrl+R**, or go to the Automation ribbon and click Interpreter – Run Script.

The Run Script dialog opens.

2 In the Run Script dialog, search the file system or use the drop-down lists to select an item that was recently used.




3 If necessary, specify additional arguments.

4 If necessary, specify a working directory. If you specify one, the Interpreter sets this as the current directory before executing the script.

**Tip**

The Python path includes the current directory automatically. You can use the drop-down list to select a directory that was recently used.

5 If you want to display the source code in the Source Code Editor, activate Open file in editor.

6 Click OK to run script execution.

**Result**

The script is executed. Standard and error outputs are displayed in the Interpreter controlbar.

## How to Specify the Python Path

**Objective**

You have to specify the Python path to tell ConfigurationDesk's Interpreter where to find the imported scripts.

**Basics**

The Python path is the list of directories Python goes through to search for modules and files.

When you import a Python module using the `import <module name>` command, the Interpreter searches the folders of the Python path for Python files of the same name with the extension PY, PYC, or PYD.

If you have developed reusable modules as libraries, you do not have to keep them in your local working folder. You can create a subfolder for them and add it to the Python path so that you can use the functions in your main script.

**Note**

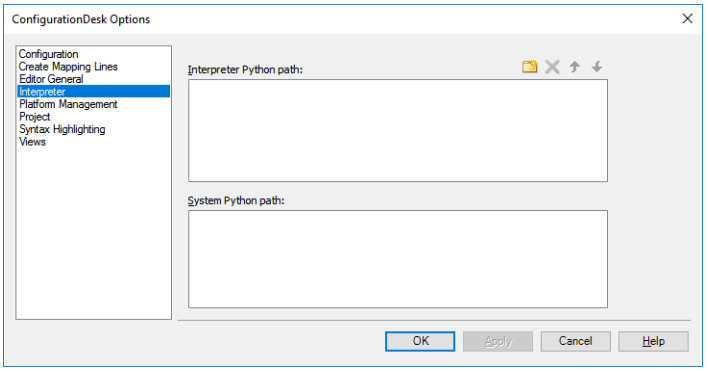
Additional Python paths specified for the Interpreter are ignored by external Python interpreters. They can only use System Python paths. System Python paths are not editable in this dialog.

**Method**

**To specify the Python path**

1 On the File ribbon, click Options.

2 Select the Interpreter page.




3 On the Interpreter page, click  to select a folder and click OK. The selected folders, such as `C:\TEMP` or `C:\OwnPythonDir`, are added to the Interpreter Python path list.

4 To change the search order of the paths, select it and click  and  to move the entry to the required position.

5 To remove a path, click an entry and click  .

6 Click OK to apply the changes.

**Result**

The specified path is appended to the list of directories which the Interpreter searches for Python modules.

# Using an External Interpreter

**Where to go from here**

Information in this section



## Basics on External Interpreters

**Introduction**

You can debug and run Python scripts in external interpreters.

**Access to ConfigurationDesk via external interpreters**

In addition to using ConfigurationDesk's Interpreter, you can also access ConfigurationDesk via external interpreters such as PythonWin or Eclipse (with the PyDev Plugin).

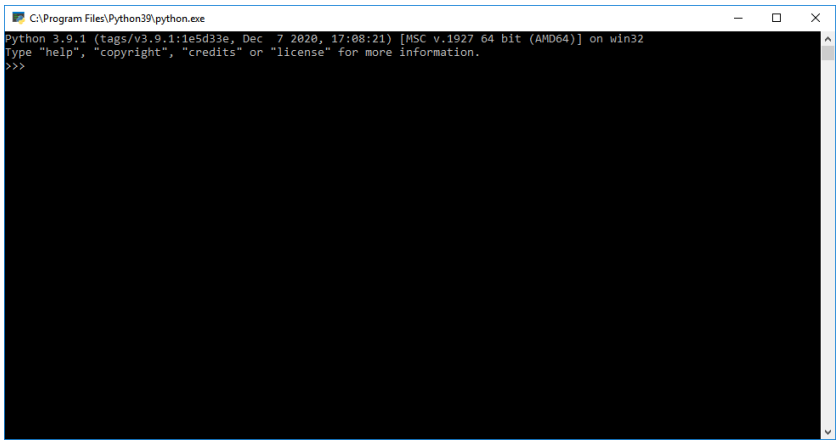
**External interpreters installed with ConfigurationDesk**

During installation of the Interpreter, the following external interpreters are also installed. The table below shows the default installation folder, provided that `C:\Program Files\` is the program files folder:

| External Interpreter | Default Installation Folder |
|---|---|
| Python.exe | `C:\Program Files\Python39` |
| Pythonw.exe | `C:\Program Files\Python39` |
| PythonWin.exe | `C:\Program Files\Python39\Lib\site-packages\pythonwin` |

**Python.exe**

`Python.exe` is a quick-to-start console. You can input single commands or run Python scripts without using command line parameters.

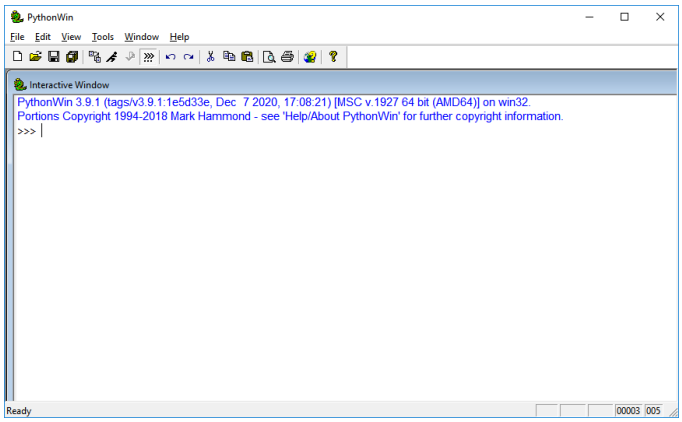



**Pythonw.exe**

`Pythonw.exe` is an interpreter without a user interface. You can run Python scripts without displaying a command window.

**PythonWin.exe**

`Pythonwin.exe` is a Windows application with an integrated context-sensitive editor and debugger. Its interactive window is similar to ConfigurationDesk's Interpreter.




**Note**

Running scripts in PythonWin might take considerably longer than in different external interpreters such as Python.exe. You should use PythonWin only to make use of its debugging features.

**PythonWin debugger** PythonWin's built-in debugger supports the usual debug features such as breakpoints, watch lists, different step modes, and post-mortem debugging.

For instructions, refer to How to Use PythonWin's Debugger on page 28.

**Related topics**

HowTos

How to Use PythonWin's Debugger....28

# How to Use PythonWin's Debugger

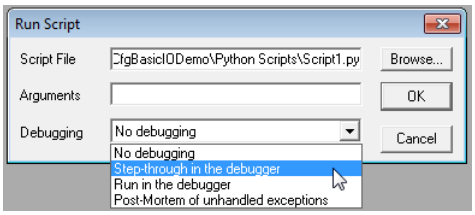
**Objective**

To debug ConfigurationDesk automation scripts, you can use PythonWin's integrated debugger.

**Method**

**To use PythonWin's debugger**

1. In PythonWin's toolbar area, click .
   The **Run Script** dialog is opened.
2. In the **Run Script** dialog, click **Debugging** and choose the debugging mode from the list.




3. Click **OK**.

**Result**

PythonWin displays the debugging results.

**Related topics**

Basics

Basics on External Interpreters....26

# Using the Source Code Editor

**Objective**

The Source Code Editor is optimized for displaying and editing Python source code.

**Where to go from here**

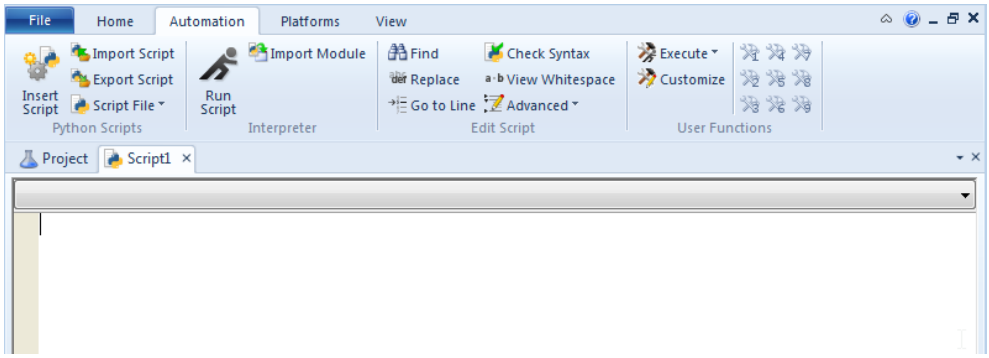
Information in this section



## Basics on the Source Code Editor

**Objective**

Python script files that you open from or create in a ConfigurationDesk project are displayed in a Source Code Editor window in the working area. There you can edit a script file and check its syntax.




**Note**

You cannot run a Python script in a Source Code Editor window. To run a Python script you can use the Run Script command in the Interpreter or on the Automation ribbon or the Run context menu command in the Project Manager.

## How to Configure the Source Code Editor

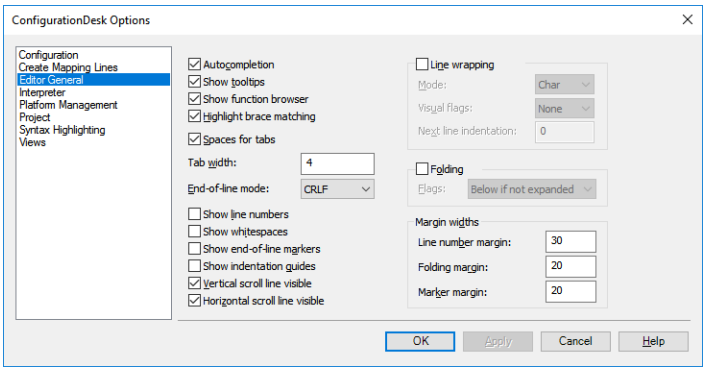
**Objective**

You can configure the Source Code Editor as required. For example, you can specify the number of spaces used for indentation.

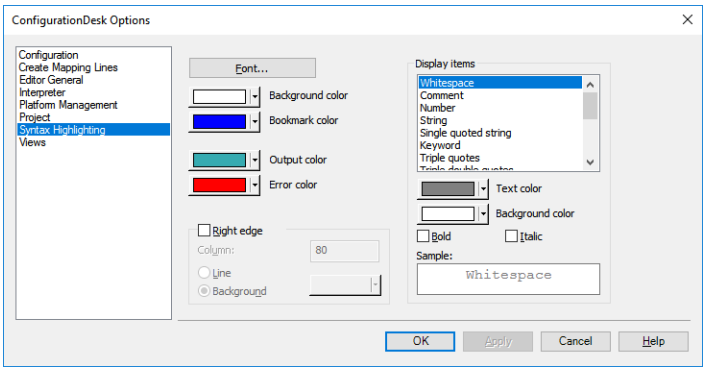
**Method**

**To configure the Source Code Editor**

1. On the File ribbon, click Options and change to the Editor General page. The ConfigurationDesk Options dialog opens.
2. On the Editor General page, configure the settings.



3 To specify the syntax coloring settings, select the **Syntax Highlighting** page.




For details on specifying the settings, refer to How to Specify Syntax Highlighting on page 18.

**Result**

You have configured the settings of the Source Code Editor.

**Related topics**

References

Editor General Page (ConfigurationDesk User Interface Reference 🕮)

## How to Create a New Python Script

**Objective**

You can insert the new Python script into a ConfigurationDesk project.

**Possible methods**

- You can create a new Python script that is inserted into the active project. Refer to Method 1 on page 31.
- You can create a new "free" Python script that is not inserted into the active project. Refer to Method 2 on page 32.

**Method 1**

**To create a new Python script that is inserted into the active project**

1 Open a project.

2 On the Automation ribbon, click Python Scripts – Insert Script.
The Insert Python Script dialog opens.

3 Enter a name for the new Python script.
The script is stored in the project's `Python Scripts` folder.

**Method 2**

**To create a new Python script that is not inserted into the active project**

1 On the Automation ribbon, click Python Scripts – Script File – New.

**Result**

A new empty Python script is opened in ConfigurationDesk's working area.

**Tip**

To change the name of a "free floating" Python script, click Script File – Save As.

**Related topics**

HowTos

How to Open a Python Script..... 32

## How to Open a Python Script

**Objective**

In the Source Code Editor, you can open an existing document or create a new one.

**Method**

**To open a Python script**

1 On the Open ribbon, click Open File, or press **Ctrl+O**.
A standard Open dialog opens.

2 Choose Py as file type and select the document you want to open.

**Result**

The selected Python script is opened in ConfigurationDesk's working area.

**Tip**

If you want to open a Python script that belongs to a project, you can double-click it in the project's `Python-Scripts` folder. For more information on inserting Python scripts into a project, refer to How to Create a New Python Script on page 31.

**Related topics**

HowTos

How to Create a New Python Script.....31

# How to Work with Selected Text

**Objective**

You can edit text in line and column mode, for example, drag it to another position.

**Method**

**To work with selected text**

1. Select the text you want to edit.

   **Tip**

   - You can select text columns by pressing **Alt** while selecting.
   - To select text with the keyboard you must press an **Arrow key+Shift**(**Shift+Alt** for column mode).

2. To increase the indent of the selected text, press **Tab**. The selected text moves to the right.
3. To decrease the indent of the selected text, press **Shift+Tab**. The selected text moves to the left.
4. To move the selected text to another position in the document, you can drag and drop it.
5. To replace the selected text, type the new text.

**Result**

You have changed the indentation level, the position, and the content of selected text.

# How to Use Bookmarks in the Source Code Editor

**Objective**

You can set bookmarks to mark specific lines in a document.

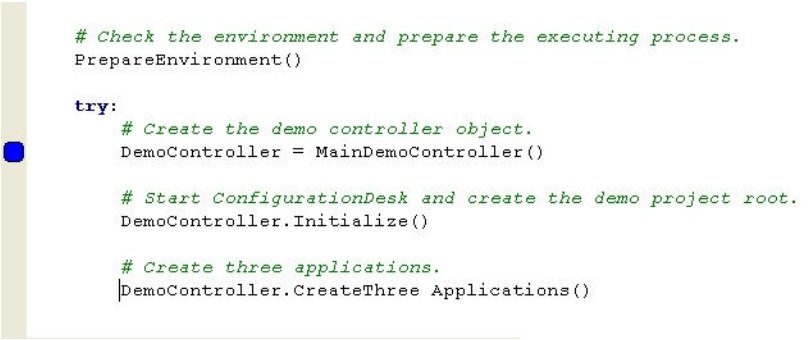
> **Note**
>
> The bookmarks you set in the Source Code Editor are cleared when you close the file.

**Method**

**To use bookmarks in the Source Code Editor**

1. Click the line where you want to set a bookmark.
2. On the Automation ribbon, click Edit Script - Advanced - Toggle Bookmark, or press **Ctrl+F2**.

   The bookmark is displayed as a blue mark in the marker margin.



```
# Check the environment and prepare the executing process.
PrepareEnvironment()

try:
    # Create the demo controller object.
    DemoController = MainDemoController()

    # Start ConfigurationDesk and create the demo project root.
    DemoController.Initialize()

    # Create three applications.
    DemoController.CreateThree Applications()
```

   If no marker margin is specified in the ConfigurationDesk Options dialog (see Editor General Page (ConfigurationDesk User Interface Reference 🕮)), the whole line is highlighted.
3. To jump to the next bookmark, go to the Automation ribbon and click Edit Script - Advanced - Next Bookmark, or press **F2**.
4. To jump to the previous bookmark, go to the Automation ribbon and click Edit Script - Advanced - Previous Bookmark, or press **Shift+F2**.
5. To clear a bookmark, click the line containing the bookmark and go to the Automation ribbon and click Edit Script - Advanced - Toggle Bookmark, or press **Ctrl+F2**.
6. To clear all bookmarks in the document, go to the Automation ribbon and click Edit Script - Advanced - Clear Bookmarks.

**Result**

You have set, selected, and cleared bookmarks in a document.

**Related topics**

References

Toggle Bookmark (ConfigurationDesk User Interface Reference 🕮)

# Basics on the Object Model in ConfigurationDesk

**Objective**

Any description of ConfigurationDesk automation needs to cover two main topics:

- The basic implementation types of the object model
- The hierarchy of the object model

**Where to go from here**

Information in this section



## The Main Types of the Object Model

**Objective**

The object model in ConfigurationDesk consists of different kinds of elements such as collections, interfaces and enumerations.

**Collection elements**

Collection elements provide access to a list of elements of the same type.

ConfigurationDesk's collection elements mostly provide methods such as the following:

- `Add`: Lets you add an element to the collection.
- `Contains`: Lets you check whether a specific element is a member of the collection.
- `Item`: Lets you access a specific element of the collection by its index, or for some collections, by its name.

ConfigurationDesk's collection elements also provide properties such as the following:

- `Count`: Provides the number of elements in the collection.

**Example** You can use the `Projects` collection to iterate through all the projects which are located under a project root. Collection names are plural.

The following listing shows how you can iterate through a `Projects` collection, printing all the names of the projects in the collection. This example works in ConfigurationDesk's Internal Interpreter.

```
# Get the currently active projects collection from the
# application object of ConfigurationDesk and
# iterates and prints out the project names
Projects = Application.Projects
for Project in Projects:
    print(Project.Name)
```

**Interfaces**

Interface elements provide access to the properties and methods of an object. Thus, a collection also implements an interface with methods and properties.

ConfigurationDesk's interface elements provide various methods such as the following:

- `Remove`: Lets you remove the element from the collection.
- `Rename`: Lets you rename the element.
- `Activate`: Lets you activate the element.

ConfigurationDesk's interface elements can provide various properties such as the following:

- `Name`: Lets you get or possibly set the name of the element.
- `Description`: Lets you get or possibly set the description text of the element.
- `Type`: Lets you get the type of the element.

**Example** You can use the `ICaActiveApplication` interface to set the description of your active application.

The following listing shows how you can set a description for the active application. This example works in ConfigurationDesk's Internal Interpreter. It assumes that a project with an application is already open.

```
# Get the active application from the application object of
# ConfigurationDesk and sets the description
ActiveApplication = Application.ActiveApplication
ActiveApplication.Description = "Throttle control test app."
```

**Event interfaces**

Event interfaces let you link the execution of Python code to the occurrence of a specific event in ConfigurationDesk. The main events in ConfigurationDesk are related to the build process of generating executable code. To automate event management, the Python script must contain a class definition for each required event source. The class definition must contain definitions for all the event handlers of the event source. For an example of using build events in automation, see the appropriate demo script.

**Enumerations**

Enumeration elements provide access to a set of named constants. Each constant can be accessed via its value or via its name. Enumerations are not a genuine element type in the Python language. When you use Python for automation, you have to import a special ConfigurationDesk Python module for access to these elements.

**Example** You can use the `ICaMainWindow` interface to set the window state of the main window of ConfigurationDesk.

The following listing shows how you can set the window state of ConfigurationDesk's main window.

| Name | Description | Value |
|---|---|---|
| Minimized | The main window is minimized. | 0 |
| Maximized | The main window is maximized. | 1 |
| Restored | The main window is restored. | 2 |

This example works in ConfigurationDesk's Internal Interpreter. It assumes that the dSPACE enums module was imported via **Import Script** on ConfigurationDesk's **Automation** ribbon or via an import statement in the script file.

```
# Dispatch the ConfigurationDesk application
from win32com.client import Dispatch
ConfigurationDeskApplication =
Dispatch("ConfigurationDesk.Application")

# Import the enum from the ConfigurationDesk python module
# "ConfigurationDeskEnums.py"
from dspace.com import Enums

# Define Enums object.
CfgDeskDeskEnums = Enums(ConfigurationDeskApplication)

# Get the main window from the application object of
# ConfigurationDesk
MainWindow = ConfigurationDeskApplication.MainWindow

# Maximize the main window
MainWindow.State = CfgDeskDeskEnums.MainWindowState.Maximized

# Restore the main window
MainWindow.State = CfgDeskDeskEnums.MainWindowState.Restored
```

## The Hierarchy of the Object Model

**Objective**

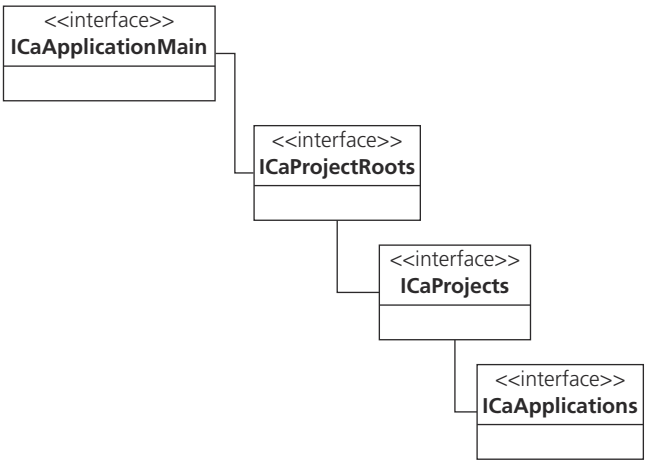
In ConfigurationDesk, the object model has a hierarchy consisting of three main parts:

- Project and application handling to create, organize, and manipulate projects and applications
- Component handling to create and configure components like the device topology, model topology, or hardware topology
- Data object handling to create or configure data objects or to connect data objects (ports) by creating links between them

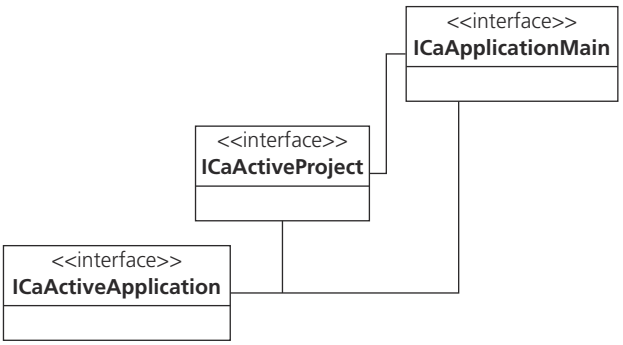
The following charts visualize only the main aspects of the object model and are not intended to show a complete design of ConfigurationDesk's automation library.

## Application handling

For automated application handling,ConfigurationDesk provides different interfaces for different tasks. Projects and applications are organized in the appropriate collections, which form a hierarchy from the one-project root collection down to the individual application collections. Every collection contains appropriate element types (`ICaProjectRoot`, `ICaProject`, `ICaApplication`).



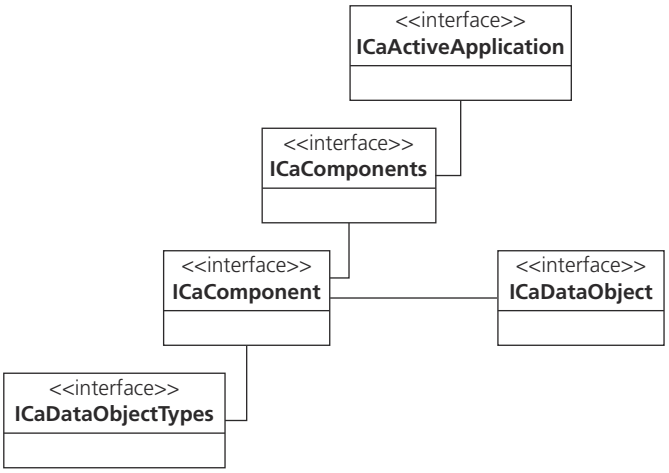

The `ICaApplicationMain` interface provides some properties for access to open projects and applications.




The active application is accessible directly via the `ICaApplicationMain` interface or from the active projects interface.

## Component handling

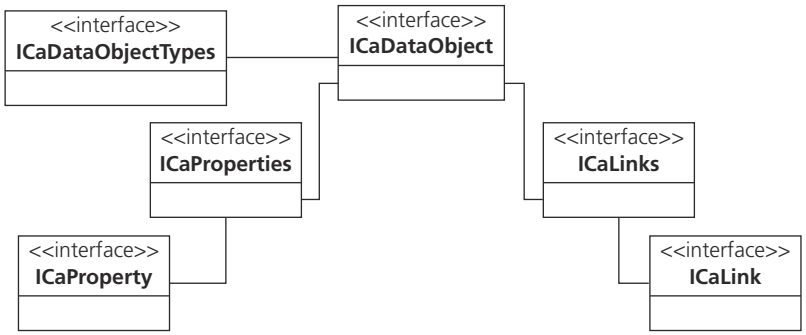
Configuring components is an important task in ConfigurationDesk, because it provides the basic data for the signal chain elements. A component is a single instance that lets you create appropriate root elements, e.g., if the component represents the device topology, you can create devices.




The `ICaDataObjectTypes` collection specifies the creatable types of an `ICaDataObject` which serves as a root object in the component.

**Data object handling**

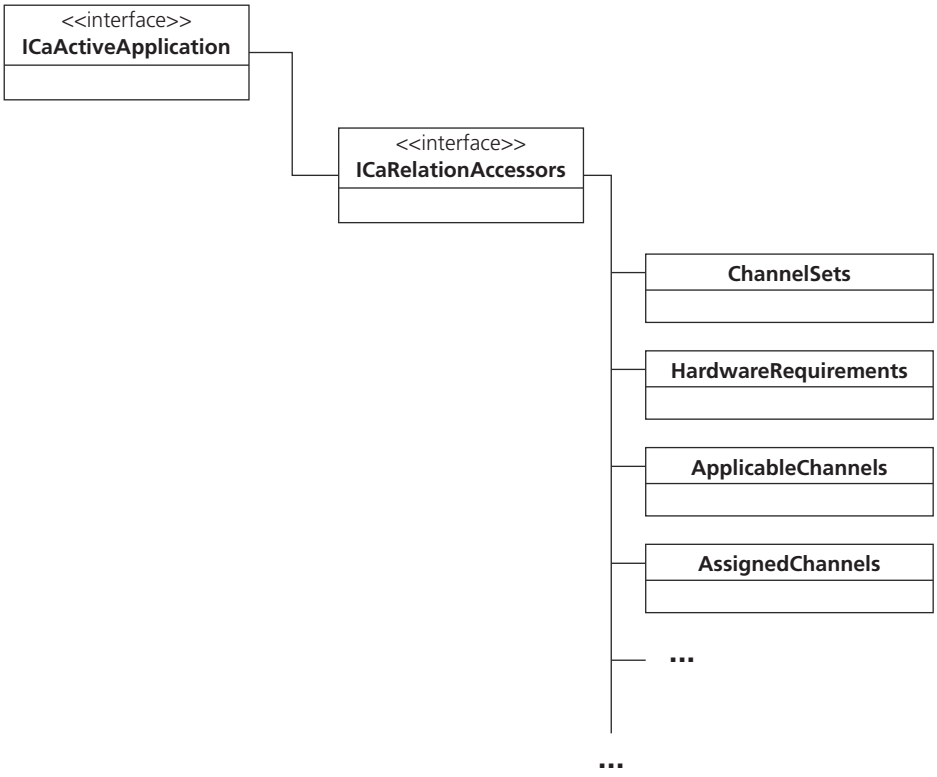
The `ICaDataObject` itself provides access to other data objects contained in it. It can also be used to create child objects, and you can query the creatable types.




Once the `ICaDataObject` is created, you can use its properties to configure it. The `ICaProperty` interface provides access to a single property, which itself has properties (such as name, value, and read-only status) that you can query. Links between data objects are accessible via the data object's `ICaLinks` collection.

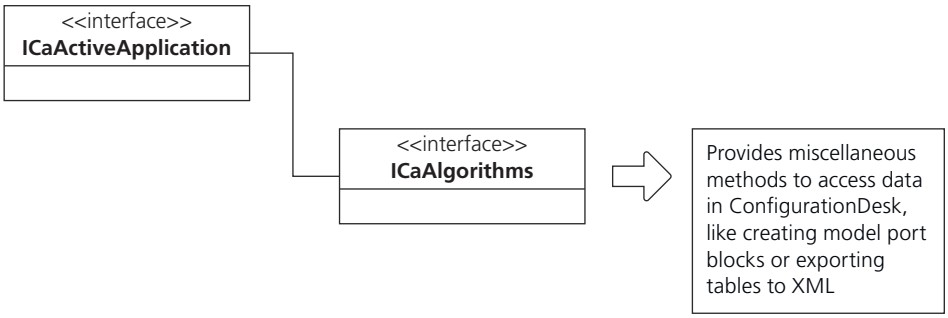
**Relations**

ConfigurationDesk automation uses the concept of relations to support configuration tasks which are not covered by the object model. When you assign channels to functions or models to application processes, you create specific relations between two or more objects. The ICaRelationAccessors interface provides access to different ICaRelationAccessor implementations which are designed for specific tasks. For example, you can use the ChannelSets, HardwareRequirements, ApplicableChannels and AssignedChannels relations to configure the hardware resource assignment for function blocks.




**Algorithms**

The ICaAlgorithms interface provides easy access to tasks which can be executed for one or more ICaDataObject by one call.




Not only configuration tasks are supported, it is also possible to export the contents of tables, the Properties Browser and the Conflicts Viewer into XML files.

# Automating ConfigurationDesk Interfaces

**Objective**

You can use Python scripts to automate different ConfigurationDesk interfaces.

The program listings in this section consist of excerpts from Python demo scripts and similar example scripts. The excerpts describe the relationships between code parts. They are not runnable by themselves. Omitted code parts are shown by an ellipsis: (...).

**Where to go from here**

Information in this section



## Basics on Automating Interfaces via Python Scripts

**Objective**

There are some points to note on structuring Python scripts and getting basic automation access that apply to all ConfigurationDesk interfaces.

**Structuring Python scripts**

The following script structure is useful for scripts that automate ConfigurationDesk:

1. Import the required modules, such as `os` or `win32com`.
2. Define functions using ConfigurationDesk's features.
3. Call the required functions in a `Main()` routine.

**Tip**

Write separate script modules to use ConfigurationDesk's features and run them when working on a project. For example, you can write separate scripts to set up a project, configure components, and work with signal chain elements.

### Getting basic access to ConfigurationDesk

To start working with ConfigurationDesk, you have to create a COM connection to ConfigurationDesk.

The following listing shows how you can open a COM connection to ConfigurationDesk using the `Dispatch` class imported from Python's `win32com.client` library.

The listing also shows the main framework of all the ConfigurationDesk automation demos: The main program calls the `ExecuteDemo` function, which uses methods of the `MainDemoController` class to start and control ConfigurationDesk.

**Note**

Initializing basic access to ConfigurationDesk via the `Dispatch` class is necessary in all the automation scripts that are described in this and the following chapters on ConfigurationDesk tool automation. It is omitted in the other scripts for the sake of readability.

```
# Import: The Dispatch class is used to create objects.
from win32com.client import Dispatch
from win32com.client import DispatchWithEvents
from dspace.com import Enums
(...)
class MainDemoController(object):
    def __init__(self):
        self.ConfigurationDeskApplication = None
        self.Enums = None
    def Initialize(self):
        self.ConfigurationDeskApplication = Dispatch("ConfigurationDesk.Application")
(...)
def ExecuteDemo():
    DemoController = None
    try:
        DemoController = MainDemoController()
        DemoController.Initialize()
(...)
# Main Program
if __name__ == '__main__':
    ExecuteDemo()
```

# Automating Project and Application Handling

**Introduction**

The main tasks of ConfigurationDesk's project management are to define projects and applications, and to open and activate them.

Unless otherwise indicated, the program listings below consist of excerpts from the `ProjectHandling.py` demo script. Keep in mind that these excerpts are not runnable by themselves.

**Note**

Initializing basic access to ConfigurationDesk via the `Dispatch` class is necessary in all the automation scripts that are described in this and the following chapters on ConfigurationDesk tool automation. It is omitted in the scripts for the sake of readability.

**Specifying a new project root directory**

The following listing shows how to create a new project root directory called `DemoRoot` in your temporary system folder. The demo scripts show a similar behavior.

```
# Import: The tempfile module is used to access the temporary system folder.
# Import: The os module is used to access the dSPACE root directory.
import tempfile
import os
# Constant: A new root path in the temporary system folder.
PROJECTROOTPATH = os.path.join(tempfile.gettempdir(), "DemoRoot")
(...)
class MainDemoController(object):
    def __init__(self):
        # The project root where most parts of the demo are executed.
        self.DemoRoot = None
    (...)
    def Initialize(self):
        # Check if the project root doesn't already exist.
        if not self.ConfigurationDeskApplication.ProjectRoots.Contains(PROJECTROOTPATH):
            # Create Demo Root Object.
            self.DemoRoot = self.ConfigurationDeskApplication.ProjectRoots.Add(PROJECTROOTPATH)
        else:
            # Get demo root.
            self.DemoRoot = self.ConfigurationDeskApplication.ProjectRoots.Item(PROJECTROOTPATH)
        # Activate new project root.
        self.DemoRoot.Activate()
```

### Creating, closing, and deleting a project

The following listing shows how to create a project with the name `NewTestProject` and then close and delete it.

```
# The name of a project used during the demo.
PROJECTNAME = "NewTestProject"
(...)
class MainDemoController(object):
    (...)
    def CreateAndDeleteNewProject(self):
        # Create a new project with the given name.
        NewProject = self.ConfigurationDeskApplication.Projects.Add(PROJECTNAME)
        (...)
        # Close project.
        NewProject.Close(False)
        # Delete project.
        self.ConfigurationDeskApplication.ActiveProjectRoot.Projects.Item(PROJECTNAME).Remove(True)
```

### Accessing an existing project

The following listing shows how to search the project root folders for the *CfgDemoTutorial* project and load it.

```
# The name of the CfgDemoTutorial project.
CFGDEMOTUTORIALPROJECTNAME = "CfgDemoTutorial"
(...)
class MainDemoController(object):
    (...)
    def LoadProject(self):
        # Check if the project exists.
        # Initialize root.
        CfgDemoTutorialRoot = None
        # Iterate over all project roots to find the root with the project searched for.
        for root in self.ConfigurationDeskApplication.ProjectRoots:
            # Check if the project was found.
            if root.Projects.Contains(CFGDEMOTUTORIALPROJECTNAME):
                # Activate project root.
                root.Activate()
                # Set FoundRoot variable to true.
                CfgDemoTutorialRoot = root
                # Stop the for loop, so the searched project root is stored in the Root variable.
                break
        # If the CfgDemoTutorial project was found, load it.
        if CfgDemoTutorialRoot:
            # Get project with given name.
            projectItem = CfgDemoTutorialRoot.Projects.Item(CFGDEMOTUTORIALPROJECTNAME)
            # Open project.
            activeProject = projectItem.Open()
```

### Creating a new application

The following listing shows how to add three new applications to a project and activate the first (for another example, see the `ProjectHandling.py` demo script).

```
# The name of a project used during the demo.
PROJECTNAME = "ApplicationHandlingDemoProject"
(...)
class MainDemoController(object):
    (...)
    def Initialize(self):
        (...)
        # Get project list.
        Projects = self.ConfigurationDeskApplication.Projects
        # Create a new project with the given name.
        self.DemoProject  = Projects.Add(PROJECTNAME)
        (...)
    def CreateThreeApplications(self):
        # Create new application 'NewApplicationOne'.
        self.DemoProject.Applications.Add("NewApplicationOne")
        # Create new applications 'NewApplicationTwo'.
        self.DemoProject.Applications.Add("NewApplicationTwo")
        # Create new applications 'NewApplicationThree'.
        self.DemoProject.Applications.Add("NewApplicationThree")
        # Activate application 'NewApplicationOne'.
        NewApplication = self.ConfigurationDeskApplication.ActiveProject.Applications.Item("NewApplicationOne")
        NewApplication.Activate()
```

## Automating Signal Chain Configuration

**Objective**

Signal chain configuration consists of many possible tasks and can be broadly subdivided into component handling, data object manipulation, and property configuration. The demo script **SignalChainHandling.py** provides examples for the most common tasks for automating ConfigurationDesk with regard to the signal chain. The **HardwareAssignment.py** demo script shows how to assign channel sets and channels to function blocks.

**Configuring topologies**

The following listing shows how to get and configure a specific component. Note that it provides only a short example and does not show all the possible aspects of configuring a component. It is assumed that a project and an application are still opened in ConfigurationDesk.

```
class MainDemoController(object):
(...)
    def ConfigureTopologies(self):
        # Get the components from the active application interface by its name
        DeviceTopology = self.ActiveApplication.Components.Item("DeviceTopology")
        ModelTopology = self.ActiveApplication.Components.Item("ModelTopology")
        Args = []
        # Add first argument: The create mode which specifies an empty topology.
        Args.append(self.Enums.DeviceTopologyCreateMode.EmptyTopology)
        # Add the name of the topology as second argument.
        # change since ConfigurationDesk 4.3 - name is only for back wards compatibility
        Args.append("NewDeviceTopology")
        # Call the configure method with the specified arguments.
        DeviceTopology.Configure("Create", Args)
        # Reset the arguments array
        Args = []
        # First add the flag for the empty mode.
        Args.append(self.Enums.ModelTopologyCreateMode.EmptyTopology)
        # Add the name for the topology - only for back wards compatibility
        Args.append("NewModelTopology")
        # Call the create method.
        ModelTopology.Configure("Create", Args)
```

**Creating data objects**

The following listing shows how to create a data object. It is assumed that a project and an application are still opened in ConfigurationDesk.

```
class MainDemoController(object):
(...)
    def AddDeviceAndHardwareDataObjectsToRepository (self):
        # Create a device in the repository / topology.
        Device = DeviceTopology.CreateRootObject(DeviceType)
        # Create a port group in the repository / topology.
        PortGroupType = None
        for Type in Device.DataObjectTypes:
        print(Type.Name)
        if Type.Name == "Port Group":
            PortGroupType = Type
        # Create the port group.
        PortGroup = Device.CreateChild(PortGroupType)
        # Create a device port in the repository / topology.
        PortType = None
        for Type in PortGroup.DataObjectTypes:
        print(Type.Name)
        if Type.Name == "Port":
            PortType = Type
        # Create port.
        Port = PortGroup.CreateChild(PortType)
        print("Name of new created port is: " + Port.Name)
```

**Setting properties**

The following listing shows how to set properties on a data object. It is assumed that a project and an application are still opened in ConfigurationDesk.

```
class MainDemoController(object):
(...)
    def SetProperties (self):
        # Get the component
        IOFunctionLib = self.ActiveApplication.Components.Item("IOFunctionLib")
        # Get the function lib as the root object
        FuncLib = IOFunctionLib.Item(0)
        # get the VoltageIn function block type
        VoltageIn = FuncLib.Item("VoltageIn")
        # Creating an instance of this function
        VoltageIn.CreateChild(VoltageIn.DataObjectTypes.Item(0))
        # Get the instance.
        VoltageInInstance = VoltageIn.Item(0)
        # Get the properties of the voltage in instance
        for Property in VoltageInInstance.Properties:
            print(Property.Name)
            print("   Value: " + str(Property.Value))
            print("...Readonly: " + str(Property.IsReadOnly))
        # try to set the InitialValueUsage property
        if VoltageInInstance.Properties.Contains("InitialValueUsage") is True:
            IniUse = VoltageInInstance.Properties.Item("InitialValueUsage")
            print("Value of Initial value usage is: " + str(IniUse.Value))
        if IniUse.Value == 1:
            IniUse.Value = 2
        else:
            IniUse.Value = 1
```

**Connecting objects**

The following listing shows how to connect objects. This is similar to creating a link graphically. It is assumed that the CfgDemoTutorial project from the dSPACE demo folder is loaded in ConfigurationDesk.

```
class MainDemoController(object):
(...)
    def ConnectObjects (self):
        # Get the component
        DeviceTopology = self.ActiveApplication.Components.Item("DeviceTopology")
        IOFunctionLib = self.ActiveApplication.Components.Item("IOFunctionLib")
        ModelTopology = self.ActiveApplication.Components.Item("ModelTopology")
        try:
            # Prepare a device ports for connecting.
            LeftDoor_D = DeviceTopology.Item(0)
            DoorLight = LeftDoor_D.Item(0)
            if DoorLight.ConnectedObjects.Count > 0:
                print("Delete the links of the DoorLight port.")
                self.ActiveApplication.DeleteLinks(DoorLight.Links)
            # Prepare a function port for connecting.
            MultiBitIn = IOFunctionLib.Item(0).Item("Multi Bit In")
            BitInDoorLightLeftF = MultiBitIn.Item(0)
            # Get the ports from the function block.
            Signal = BitInDoorLightLeftF.Item(0).Item(0).Item("Signal")
            Value = BitInDoorLightLeftF.Item(1).Item("Value")
            # Delete the links of the Reference port.
            self.ActiveApplication.DeleteLinks(Value.Links)
            # Prepare a model port for connecting.
            LeftDoor_M = ModelTopology.Item(0).Item(0).Item(0)
            BitInDoorLightLeft_M = LeftDoor_M.Item(0)
            # Get the port from the model block.
            DigitalValue = BitInDoorLightLeft_M.Item(0)
            # Now connect the ports
            self.ActiveApplication.ConnectObjects(DoorLight, Signal)
            self.ActiveApplication.ConnectObjects(Value, DigitalValue)
        except:
            return
```

**Working with working views**

The following listing shows how to work with working views and working view groups. It is assumed that a project and an application are still opened in ConfigurationDesk.

```
class MainDemoController(object):
(...)
    def WorkWithWorkingViews (self):
        # Get the working views collection from the active application
        WorkingViews = self.ActiveApplication.WorkingViews
        for WorkingView in WorkingViews:
            print("WorkingView name is: " + WorkingView.Name)
        # Get the global working view which contains all ports of the application
        Global = WorkingViews.Item("Global")
        print("Number of items in the global working view is: " + str(Global.Count))
        # Add a working view group under the root item
        WorkingViews.AddWorkingViewGroup("", "NewGroup")
        # Add another group under the new created group
        WorkingViews.AddWorkingViewGroup("NewGroup", "NextNewGroup")
        # Add a working view in the last created group
        View = WorkingViews.Add("NewGroup\NextNewGroup", "NewView")
        # Add another working view in this group
        WorkingViews.Add("NewGroup\NextNewGroup", "NextNewView")
        # Move the view before to the group above
        WorkingViews.Move(View.FullName, "NewGroup")
        # Now move it to the root
        WorkingViews.Move(View.FullName, "")
```

### Using relations to assign a channel set

The following listing shows how to use the ICaRelationAccessors interface to assign a channel set to a Voltage Out function. It is assumed that the CfgDemoTutorial project from the dSPACE demo folder is loaded in ConfigurationDesk.

Channels are assigned to functions in four steps:

1. Assign a channel set to the electrical interface (internal name: signal conditioning) part of the required function instance.
2. Get the hardware requirements from this part.
3. Get the applicable channels which are provided for this object.
4. Assign a channel to the electrical interface (internal name: signal conditioning) part.

To see all the automation steps for assigning a channel to a function, read the HardwareAssignment.py demo script in the demo folder.

```
class MainDemoController(object):
(...)
    def AssignChannelSets (self):
        # Get the RelationAccessor which is used for channel sets
        ChannelSetRelation = self.ActiveApplication.Relations.Item("ChannelSets")
        # Get the function topology to create an instance of the VoltageOut function
        FunctionTopology = self.ActiveApplication.Components.Item("IOFunctionLib")
        FunctionLibrary = FunctionTopology.Item(0)
        # create the instance
        VoltageOut = FunctionLibrary.Item("Voltage Out")
        VoltageOutInstance = VoltageOut.CreateChild(VoltageOut.DataObjectTypes.Item(0))
        VoltageOutInstance.IsInApplication = True
        # Get the signal conditioning part
        SignalCond = VoltageOutInstance.Item(0)
        # Get the currently available channel sets with the RelationAccessor
        AvailableSets = ChannelSetRelation.GetElements(SignalCond)
        # Get the first available channel set for the assignment
        ChSet = AvailableSets.Item(0)
        # Create the arguments array for the relation call
        Args = []
        Args.append(ChSet)
        # Assign the channel set to the signal conditioning part of the function instance
        ChannelSetRelation.SetElements(SignalCond, Args)
```

### Using algorithms to create model port blocks

The following listing shows how to use the ICaAlgorithms interface to create a suitable model port block for a Voltage In function. It is assumed that a project and an application are still opened in ConfigurationDesk.

```
class MainDemoController(object):
(...)
    def ExecuteAlgorithmsForDataObjects (self):
        IOFunctionLib = self.ActiveApplication.Components.Item("IOFunctionLib")
        # Get the VoltageIn I/O function from the topology.
        VoltageIn = IOFunctionLib.Item(0).Item("Voltage In")
        # Create a child to instantiate the function
        VoltageIn.CreateChild(VoltageIn.DataObjectTypes.Item(0))
        VoltageInInstance = VoltageIn.Item(0)
        # Create a list which contains the blocks to create model ports for.
        FunctionBlocks = []
        FunctionBlocks.append(VoltageInInstance)
        # Call the algorithms function to create the model port block
        self.ActiveApplication.Algorithms.CreateSuitableModelPortBlock(FunctionBlocks)
```

# Automating Platform Management

**Objective**

The main tasks of ConfigurationDesk's platform management are to show registered hardware and to load or unload real time applications to or from the connected hardware.

**Note**

Only the basic initialization of platform management automation is available with ConfigurationDesk. To access the complete platform management API, you must install a product set containing the Platform API Package.

**Registering hardware**

The program listings below consist of excerpts from the `PlatformHandling.py` demo script.

The following listing shows the initialization of the MainDemoController.

```
class MainDemoController(object):
    def __init__(self):
        # Initialize the ConfigurationDesk application object.
        self.ConfigurationDeskApplication = None
        # Initialize IP addresses.
        self.IPAddress1 = None
        self.IPAddress2 = None
        # Initialize the hardware system object.
        self.ScalexioHardware = None
        # Initialize the path to the "*.rta" file which contains the real-time application.
        self.PathToRealTimeApplicationFile = None
        # Initialize the real-time application object.
        self.RealTimeApplication = None
```

The following listing shows you how to register a SCALEXIO multi-processing-unit system for ConfigurationDesk with the API Version 2.0.

```
def RegisterHardwareAPIVersion2(self):
    # Local variable for convenient access.
    PlatformManagement = self.ConfigurationDeskApplication.PlatformManagement
    # Activate the platform automation API version 2
    PlatformManagement.PlatformAutomationAPIVersion = 2
    # It is a precondition that the hardware with the used IP address was not already
    # registered before - therefore we remove all registered hardware
    PlatformManagement.RecentPlatformConfiguration.RemoveAll()
    # Create a registration info object with the desired type (SCALEXIO).
    RegistrationInfo = PlatformManagement.CreatePlatformRegistrationInfo(self.Enums.PlatformType.SCALEXIO)
    # Add two registration infos for the multi PU system
    # and set the IP address of the system to connect.
    RegInfo1 = RegistrationInfo.RegistrationInfos.Add()
    RegInfo1.IPAddress = self.IPAddress1
    RegInfo2 = RegistrationInfo.RegistrationInfos.Add()
    RegInfo2.IPAddress = self.IPAddress2
    # Register the hardware
    Platform = PlatformManagement.RegisterPlatform(RegistrationInfo)
    #  print out the names of the processing units
    for ProcessingUnit in Platform.ProcessingUnits:
        print(ProcessingUnit.DisplayName)
    # remove all registered platforms
    PlatformManagement.RecentPlatformConfiguration.RemoveAll()
```

### Scanning registered hardware after program start

When starting ConfigurationDesk, the software scans the connected network to find registered dSPACE real-time hardware. If you start ConfigurationDesk automated via Python script, the scan process is skipped automatically because it might take too long and cause a timeout. This would abort the execution of the script.

To search for the connected platforms after startup via tool automation, call the RefreshPlatformConfiguration method.

```
Application.PlatformManagement.RefreshPlatformConfiguration()
```

Now ConfigurationDesk searches for registered hardware in the connected network on startup.

# Automating Bus Manager Features

**Introduction**

The ConfigurationDesk automation interface provides some specific elements for automating features of the Bus Manager.

**Where to go from here**

Information in this section



## Basics on Automating Bus Manager Features

**Introduction**

You can automate Bus Manager features via various interfaces, and easily access elements of the Bus Manager.

**Automating Bus Manager features**

You can automate Bus Manager features via the same interfaces that are used to automate ConfigurationDesk's signal chain configuration. Additionally, the ICaRelations <<Collection>> and ICaComponent <<Collection>> interfaces provide Bus Manager-specific relations and operations.

The Bus Manager-specific relations and operations let you, for example:

- Add bus configurations and communication matrices to a ConfigurationDesk application.
- Assign and remove communication matrix elements to and from bus configurations.
- Configure bus configurations.
- Specify user-defined settings for communication matrix elements and export overviews of the modified communication matrix elements.
- Replace the currently assigned communication matrix of a bus configuration by another communication matrix.

- Generate bus simulation containers that contain bus configurations of the ConfigurationDesk application.

### Accessing Bus Manager elements

ConfigurationDesk's automation interface lets you access elements of bus configurations and communication matrices. You can access the communication matrix elements sorted by ECUs or clusters. To access Bus Manager elements, you can navigate through the respective hierarchy tree or directly access specific elements via XPath expressions. The XPath expressions must comply with XPath 1.0. Other XPath versions are not supported.

### Best practices for accessing Bus Manager elements via XPath

When you access Bus Manager elements via XPath, it is recommended to not access the elements by their absolute position in a communication matrix or a bus configuration. The absolute position of elements in communication matrices and bus configurations might change between dSPACE Releases. If it does and you access elements by their absolute position, you must adapt the relevant automation scripts. To avoid this, access elements by other criteria, for example, by their name or by their relative position to higher-level or lower-level elements.

**Examples** Avoid accessing elements via XPath as follows:

```
# Accessing the second ECU
'//BusEcu[2]'

# Accessing the third BusContainerIPdu of the first ECU
'//BusEcu[1]/BusContainerIPdu[3]'
```

Instead, access elements via XPath as follows:

```
# Accessing all elements with the 'RX' direction below all
# ECUs with the name 'Node3'
'//BusEcu[@Name = "Node3"]/*[@Direction = "RX"]'

# Accessing all ISignals whose name contains '_2' below all ISignal IPDUs
# with the 'RX' direction
'//BusISignalIPdu[@Direction = "RX"]/BusISignal[contains(@Name, "_2")]'

# Accessing all elements that are two levels below all LIN bus systems
'//BusSystemLin/*/*'
```

### Adding bus configuration features to bus configurations

One way to configure bus configurations is to add bus configuration features to elements of bus configurations (e.g., the ISignal Value feature to an ISignal or the PDU Trigger feature to a PDU).

You can add a bus configuration feature to a bus configuration element only if the feature is available for the specific element. The availability of bus configuration features depends on various factors, e.g., on the element type of a bus configuration element or the bus configuration part the element is assigned to.

To add a bus configuration feature, you must know the feature's role name. Via the ICaComponent <<Collection>> interface, you can get a list of the role

names of bus configuration features that are available for a specific bus configuration element, and add features to the element.

**Tip**

The syntax of most bus configuration feature role names is:

- `Bus<BusConfigurationFeatureName>Access` for features of bus configuration elements that are assigned to the Simulated ECUs bus configuration part
- `Bus<BusConfigurationFeatureName>Inspection` for features of bus configuration elements that are assigned to the Inspection bus configuration part
- `Bus<BusConfigurationFeatureName>Manipulation` for features of bus configuration elements that are assigned to the Manipulation bus configuration part

For example, the role name of the ISignal Value feature is `BusISignalValueAccess` if the related ISignal is assigned to the Simulated ECUs part and `BusISignalValueInspection` if the ISignal is assigned to the Inspection part of a bus configuration.

The Element Type column of bus configuration tables indicates the role names. For example, the element type of the PDU Trigger feature is Bus PDU Trigger Access. The features's role name is `BusPduTriggerAccess`.

# Accessing Communication Matrix Elements Sorted by Clusters via XPath

**Introduction**

ConfigurationDesk's automation interface provides a relation for accessing communication matrix elements sorted by clusters. By using this relation, you can access these elements via XPath expressions.

**Relation for accessing communication matrix elements sorted by clusters**

You can use the `CommunicationMatricesByClusters` relation of the ICaRelations <<Collection>> on page 160 interface to access communication matrix elements sorted by clusters.

**XPath expressions**

The following table provides an overview of the XPath expressions for accessing communication matrix elements sorted by clusters. The slash (/) is part of the XPath expression.

For more information on the individual elements, refer to Elements and Symbols of Bus Configurations and Communication Matrices (ConfigurationDesk User Interface Reference 🕮).

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| Bus Communication Matrix | `/BusCommunicationMatrix` |  | - | ▪ Bus System CAN<br>▪ Bus System LIN |

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| Bus System CAN | `/BusSystemCan` |  | Bus Communication Matrix | Bus CAN Communication Cluster |
| Bus System LIN | `/BusSystemLin` |  | Bus Communication Matrix | Bus LIN Communication Cluster |
| Bus CAN Communication Cluster | `/BusCanCommunicationCluster` |  | Bus System CAN | Bus Network Node |
| Bus LIN Communication Cluster | `/BusLinCommunicationCluster` |  | Bus System LIN | Bus Network Node |
| Bus Network Node | `/BusNetworkNode` |  | ▪ Bus CAN Communication Cluster<br>▪ Bus LIN Communication Cluster | ▪ Bus CAN Physical Channel<br>▪ Bus LIN Physical Channel |
| Bus CAN Physical Channel | `/BusCanPhysicalChannel` |  | Bus Network Node | ▪ Bus Container IPDU<br>▪ Bus Multiplexed IPDU<br>▪ Bus Secured IPDU<br>▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus General-Purpose PDU<br>▪ Bus DCM IPDU<br>▪ Bus NMPDU<br>▪ Bus NPDU<br>▪ Bus User-Defined IPDU<br>▪ Bus User-Defined PDU |
| Bus LIN Physical Channel | `/BusLinPhysicalChannel` |  | Bus Network Node | ▪ Bus ISignal IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus General-Purpose PDU<br>▪ Bus DCM IPDU<br>▪ Bus NMPDU<br>▪ Bus NPDU<br>▪ Bus User-Defined IPDU<br>▪ Bus User-Defined PDU |
| Bus Container IPDU | `/BusContainerIPdu` | ▪  (TX)<br>▪  (RX) | ▪ Bus CAN Physical Channel<br>▪ Bus Secured IPDU | ▪ Bus Multiplexed IPDU<br>▪ Bus Secured IPDU<br>▪ Bus ISignal IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus DCM IPDU<br>▪ Bus User-Defined IPDU |
| Bus Multiplexed IPDU | `/BusMultiplexedIPdu` | ▪  (TX)<br>▪  (RX) | ▪ Bus CAN Physical Channel<br>▪ Bus Container IPDU<br>▪ Bus Secured IPDU | Bus ISignal IPDU |
| Bus Secured IPDU | `/BusSecuredIPdu` | ▪  (TX) | ▪ Bus CAN Physical Channel | ▪ Bus Container IPDU |

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| | | ▪  (RX) | ▪ Bus Container IPDU | ▪ Bus Multiplexed IPDU<br>▪ Bus ISignal IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus DCM IPDU<br>▪ Bus User-Defined IPDU |
| Bus ISignal IPDU | `/BusISignalIPdu` | ▪  (TX)<br>▪  (RX) | ▪ Bus CAN Physical Channel<br>▪ Bus LIN Physical Channel<br>▪ Bus Container IPDU<br>▪ Bus Multiplexed IPDU<br>▪ Bus Secured IPDU | ▪ Bus ISignal Group<br>▪ Bus ISignal |
| Bus Extended Multiplexed IPDU | `/BusExtendedMultiplexedIPdu` | ▪  (TX)<br>▪  (RX) | Bus CAN Physical Channel | Bus ISignal |
| Bus General-Purpose IPDU | `/BusGeneralPurposeIPdu` | ▪  (TX)<br>▪  (RX) | ▪ Bus CAN Physical Channel<br>▪ Bus LIN Physical Channel<br>▪ Bus Container IPDU<br>▪ Bus Secured IPDU | - |
| Bus General-Purpose PDU | `/BusGeneralPurposePdu` | ▪  (TX)<br>▪  (RX) | ▪ Bus CAN Physical Channel<br>▪ Bus LIN Physical Channel | Bus Global Time Domain |
| Bus DCM IPDU | `/BusDcmIPdu` | ▪  (TX)<br>▪  (RX) | ▪ Bus CAN Physical Channel<br>▪ Bus LIN Physical Channel<br>▪ Bus Container IPDU<br>▪ Bus Secured IPDU | - |
| Bus NMPDU | `/BusNmPdu` | ▪  (TX)<br>▪  (RX) | ▪ Bus CAN Physical Channel<br>▪ Bus LIN Physical Channel | ▪ Bus ISignal Group<br>▪ Bus ISignal |
| Bus NPDU | `/BusNPdu` | ▪  (TX)<br>▪  (RX) | ▪ Bus CAN Physical Channel<br>▪ Bus LIN Physical Channel | - |
| Bus User-Defined IPDU | `/BusUserDefinedIPdu` | ▪  (TX)<br>▪  (RX) | ▪ Bus CAN Physical Channel<br>▪ Bus LIN Physical Channel<br>▪ Bus Container IPDU<br>▪ Bus Secured IPDU | - |
| Bus User-Defined PDU | `/BusUserDefinedPdu` | ▪  (TX)<br>▪  (RX) | ▪ Bus CAN Physical Channel<br>▪ Bus LIN Physical Channel | - |
| Bus ISignal Group | `/BusISignalGroup` | ▪  (TX)<br>▪  (RX) | ▪ Bus ISignal IPDU<br>▪ Bus NMPDU | Bus ISignal |
| Bus ISignal | `/BusISignal` | ▪  (TX)<br>▪  (RX) | ▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus NMPDU<br>▪ Bus ISignal Group | - |
| Bus Global Time Domain | `/BusGlobalTimeDomain` | ▪  (TX)<br>▪  (RX) | Bus General-Purpose PDU | - |

You can also access the properties of communication matrix elements via XPath expressions. For an overview of the valid expressions, refer to Accessing Element Properties via XPath on page 80.

**Examples**

The following listing provides short examples for accessing specific elements of a communication matrix sorted by clusters.

```
# Accessing all BusCanCommunicationClusters in all CAN BusSystems in all BusCommunicationMatrices
'/BusCommunicationMatrix/BusSystemCan/BusCanCommunicationCluster'

# Accessing all BusCanPhysicalChannels in all BusNetworkNodes whose name contains 'door' in all
# BusCanCommunicationClusters in all CAN BusSystems in all BusCommunicationMatrices
'/BusCommunicationMatrix/BusSystemCan/BusCanCommunicationCluster/BusNetworkNode[contains(@Name,
"door")]/BusCanPhysicalChannel'

# Accessing all CAN BusSystems below all root elements
'/*/BusSystemCan'

# Accessing all LIN BusSystems
'//BusSystemLin'

# Accessing all elements that are two levels below all LIN BusSystems
'//BusSystemLin/*/*'

# Accessing all elements with the 'RX' direction and that are two levels below all BusNetworkNodes
# with the name 'Node3'
'//BusNetworkNode[@Name = "Node3"]/*/*[@Direction = "RX"]'

# Accessing all BusISignals whose name contains '_2' below all BusISignalIPdus with the 'RX' direction
'//BusISignalIPdu[@Direction = "RX"]/BusISignal[contains(@Name, "_2")]'
```

## Accessing Communication Matrix Elements Sorted by ECUs via XPath

**Introduction**

ConfigurationDesk's automation interface provides a relation for accessing communication matrix elements sorted by ECUs. By using this relation, you can access these elements via XPath expressions.

**Relation for accessing communication matrix elements sorted by ECUs**

You can use the `CommunicationMatricesByEcus` relation of the ICaRelations <<Collection>> on page 160 interface to access communication matrix elements sorted by ECUs.

**XPath expressions**

The following table provides an overview of the XPath expressions for accessing communication matrix elements sorted by ECUs. The slash (`/`) is part of the XPath expression.

For more information on the individual elements, refer to Elements and Symbols of Bus Configurations and Communication Matrices (ConfigurationDesk User Interface Reference 🕮).

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| Bus Communication Matrix | `/BusCommunicationMatrix` |  | - | Bus ECU |
| Bus ECU | `/BusEcu` |  | Bus Communication Matrix | ▪ Bus Container IPDU<br>▪ Bus Multiplexed IPDU<br>▪ Bus Secured IPDU<br>▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus General-Purpose PDU<br>▪ Bus DCM IPDU<br>▪ Bus NMPDU<br>▪ Bus NPDU<br>▪ Bus User-Defined IPDU<br>▪ Bus User-Defined PDU |
| Bus Container IPDU | `/BusContainerIPdu` | ▪  (TX)<br>▪  (RX) | ▪ Bus ECU<br>▪ Bus Secured IPDU | ▪ Bus Multiplexed IPDU<br>▪ Bus Secured IPDU<br>▪ Bus ISignal IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus DCM IPDU<br>▪ Bus User-Defined IPDU |
| Bus Multiplexed IPDU | `/BusMultiplexedIPdu` | ▪  (TX)<br>▪  (RX) | ▪ Bus ECU<br>▪ Bus Container IPDU<br>▪ Bus Secured IPDU | Bus ISignal IPDU |
| Bus Secured IPDU | `/BusSecuredIPdu` | ▪  (TX)<br>▪  (RX) | ▪ Bus ECU<br>▪ Bus Container IPDU | ▪ Bus Container IPDU<br>▪ Bus Multiplexed IPDU<br>▪ Bus ISignal IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus DCM IPDU<br>▪ Bus User-Defined IPDU |
| Bus ISignal IPDU | `/BusISignalIPdu` | ▪  (TX)<br>▪  (RX) | ▪ Bus ECU<br>▪ Bus Container IPDU<br>▪ Bus Multiplexed IPDU<br>▪ Bus Secured IPDU | ▪ Bus ISignal Group<br>▪ Bus ISignal |
| Bus Extended Multiplexed IPDU | `/BusExtendedMultiplexedIPdu` | ▪  (TX)<br>▪  (RX) | Bus ECU | Bus ISignal |
| Bus General-Purpose IPDU | `/BusGeneralPurposeIPdu` | ▪  (TX)<br>▪  (RX) | ▪ Bus ECU<br>▪ Bus Container IPDU<br>▪ Bus Secured IPDU | - |
| Bus General-Purpose PDU | `/BusGeneralPurposePdu` | ▪  (TX)<br>▪  (RX) | Bus ECU | Bus Global Time Domain |

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| Bus DCM IPDU | `/BusDcmIPdu` | ▪ (TX)<br>▪ (RX) | ▪ Bus ECU<br>▪ Bus Container IPDU<br>▪ Bus Secured IPDU | - |
| Bus NMPDU | `/BusNmPdu` | ▪ (TX)<br>▪ (RX) | Bus ECU | ▪ Bus ISignal Group<br>▪ Bus ISignal |
| Bus NPDU | `/BusNPdu` | ▪ (TX)<br>▪ (RX) | Bus ECU | - |
| Bus User-Defined IPDU | `/BusUserDefinedIPdu` | ▪ (TX)<br>▪ (RX) | ▪ Bus ECU<br>▪ Bus Container IPDU<br>▪ Bus Secured IPDU | - |
| Bus User-Defined PDU | `/BusUserDefinedPdu` | ▪ (TX)<br>▪ (RX) | Bus ECU | - |
| Bus ISignal Group | `/BusISignalGroup` | ▪ (TX)<br>▪ (RX) | ▪ Bus ISignal IPDU<br>▪ Bus NMPDU | Bus ISignal |
| Bus ISignal | `/BusISignal` | ▪ (TX)<br>▪ (RX) | ▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus NMPDU<br>▪ Bus ISignal Group | - |
| Bus Global Time Domain | `/BusGlobalTimeDomain` | ▪ (TX)<br>▪ (RX) | Bus General-Purpose PDU | - |

You can also access the properties of communication matrix elements via XPath expressions. For an overview of the valid expressions, refer to Accessing Element Properties via XPath on page 80.

**Examples**

The following listing provides short examples for accessing specific elements of a communication matrix sorted by ECUs.

```
# Accessing the BusCommunicationMatrix with the name 'autosar_321_CanLinFlexRay_01'
'/BusCommunicationMatrix[@Name = "autosar_321_CanLinFlexRay_01"]'

# Accessing all elements one level below the BusCommunicationMatrix
'//BusCommunicationMatrix/*'

# Accessing all BusEcus whose name contains 'door'
'//BusEcu[contains(@Name, "door")]'

# Accessing all elements with the 'RX' direction below all BusEcus with the name 'Node3'
'//BusEcu[@Name = "Node3"]/*[@Direction = "RX"]'

# Accessing all BusISignals whose name contains '_2' in all BusISignalIPdus with length <= 16 and the 'TX' direction
'//BusISignalIPdu[@length <= 16 and @Direction = "TX"]/BusISignal[contains(@Name, "_2")]'
```

# Accessing Bus Configuration Elements via XPath

**Introduction**

ConfigurationDesk's automation interface provides a relation for accessing bus configuration elements. By using this relation, you can access these elements via XPath expressions.

**Relation for accessing bus configuration elements**

You can use the `BusConfigurations` relation of the ICaRelations <<Collection>> on page 160 interface to access bus configuration elements.

**XPath expressions**

The following tables provide overviews of the XPath expressions for accessing bus configuration elements. The slash (`/`) is part of the XPath expression.

You can also access the properties of bus configuration elements via XPath expressions. For an overview of the valid expressions, refer to Accessing Element Properties via XPath on page 80.

For more information on the individual elements, refer to Elements and Symbols of Bus Configurations and Communication Matrices (ConfigurationDesk User Interface Reference).

**Accessing the Bus Access Requests bus configuration part** The following table provides an overview of the XPath expressions for accessing a bus configuration and the elements that can be available for its Bus Access Requests part.

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| Bus Configuration | `/BusConfiguration` |  | - | Bus Configuration Part Bus Access Requests |
| Bus Configuration Part Bus Access Requests | `/BusConfigurationPartBusAccessRequests` |  | Bus Configuration | ▪ Bus Communication Matrix<br>▪ Bus Frame Capture<br>▪ Bus Frame Gateway |
| Bus Communication Matrix | `/BusCommunicationMatrix` |  | Bus Configuration Part Bus Access Requests | ▪ Bus System CAN<br>▪ Bus System LIN |
| Bus Frame Capture | `/BusFrameCapture` |  | Bus Configuration Part Bus Access Requests | Bus System CAN |
| Bus Frame Gateway | `/BusFrameGateway` |  | Bus Configuration Part Bus Access Requests | Bus System CAN |
| Bus System CAN | `/BusSystemCan` |  | ▪ Bus Communication Matrix<br>▪ Bus Frame Capture<br>▪ Bus Frame Gateway | ▪ Bus CAN Communication Cluster<br>▪ Bus CAN Frame Capture Cluster<br>▪ Bus CAN Frame Gateway Cluster |
| Bus System LIN | `/BusSystemLin` |  | Bus Communication Matrix | Bus LIN Communication Cluster |

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| Bus CAN Communication Cluster | `/BusCanCommunicationCluster` |  | Bus System CAN | ▪ Bus Access Request Simulated ECUs<br>▪ Bus Access Request Inspection<br>▪ Bus Access Request Manipulation |
| Bus CAN Frame Capture Cluster | `/BusCanFrameCaptureCluster` |  | Bus System CAN | Bus Access Request Inspection |
| Bus CAN Frame Gateway Cluster | `/BusCanFrameGatewayCluster` |  | Bus System CAN | Bus Access Request Gateways |
| Bus LIN Communication Cluster | `/BusLinCommunicationCluster` |  | Bus System LIN | ▪ Bus Access Request Simulated ECUs<br>▪ Bus Access Request Inspection<br>▪ Bus Access Request Manipulation |
| Bus Access Request Simulated ECUs | `/BusAccessRequestSimulatedEcus` |  | ▪ Bus CAN Communication Cluster<br>▪ Bus LIN Communication Cluster | Function Block |
| Bus Access Request Inspection | `/BusAccessRequestInspection` |  | ▪ Bus CAN Communication Cluster<br>▪ Bus CAN Frame Capture Cluster<br>▪ Bus LIN Communication Cluster | Function Block |
| Bus Access Request Manipulation | `/BusAccessRequestManipulation` |  | ▪ Bus CAN Communication Cluster<br>▪ Bus LIN Communication Cluster | Function Block |
| Bus Access Request Gateways | `/BusAccessRequestGateways` |  | Bus CAN Frame Gateway Cluster | Function Block |
| Function Block | `/FunctionBlock` |  | ▪ Bus Access Request Simulated ECUs<br>▪ Bus Access Request Inspection<br>▪ Bus Access Request Manipulation<br>▪ Bus Access Request Gateways | - |

**Accessing the Simulated ECUs bus configuration part** The following table provides an overview of the XPath expressions for accessing a bus configuration and the elements that can be available for its Simulated ECUs part.

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| Bus Configuration | `/BusConfiguration` |  | - | ▪ Bus Configuration Enable Global<br>▪ Bus Configuration Part Simulated ECUs |

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| Bus Configuration Enable Global | `/BusConfigurationEnableGlobal` |  | Bus Configuration | Function Port |
| Bus Configuration Part Simulated ECUs | `/BusConfigurationPartSimulatedEcus` |  | Bus Configuration | Bus Communication Matrix |
| Bus Communication Matrix | `/BusCommunicationMatrix` |  | Bus Configuration Part Simulated ECUs | Bus ECU |
| Bus ECU | `/BusEcu` |  | Bus Communication Matrix | ▪ Bus CAN Communication Controller<br>▪ Bus LIN Communication Controller<br>▪ Bus Container IPDU<br>▪ Bus Multiplexed IPDU<br>▪ Bus Secured IPDU<br>▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus General-Purpose PDU<br>▪ Bus DCM IPDU<br>▪ Bus NMPDU<br>▪ Bus NPDU<br>▪ Bus User-Defined IPDU<br>▪ Bus User-Defined PDU |
| Bus CAN Communication Controller | `/BusCanCommunicationController` |  | Bus ECU | ▪ Bus Communication Controller Enable Access<br>▪ Bus J1939 Network Management Enable Access |
| Bus LIN Communication Controller | `/BusLinCommunicationController` | ▪  (LIN master)<br>▪  (LIN slave) | Bus ECU | ▪ Bus Communication Controller Enable Access<br>▪ Bus Communication Controller LIN Schedule Table Access<br>▪ Bus Communication Controller LIN Wake-Up Access |
| Bus Container IPDU | `/BusContainerIPdu` | ▪  (TX)<br>▪  (RX) | Bus ECU | ▪ Bus Frame Access<br>▪ Bus PDU Raw Data Access<br>▪ Bus PDU RX Interrupt Access<br>▪ Bus PDU RX Status Access |
| Bus Multiplexed IPDU | `/BusMultiplexedIPdu` | ▪  (TX)<br>▪  (RX) | Bus ECU | ▪ Bus Frame Access<br>▪ Bus PDU Raw Data Access<br>▪ Bus PDU RX Interrupt Access<br>▪ Bus PDU RX Status Access |
| Bus Secured IPDU | `/BusSecuredIPdu` | ▪  (TX)<br>▪  (RX) | Bus ECU | ▪ Bus Frame Access<br>▪ Bus PDU Raw Data Access<br>▪ Bus PDU RX Status Access |

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| | | | | ▪ Bus PDU SecOC Access<br>▪ Bus PDU User Code Access |
| Bus ISignal IPDU | `/BusISignalIPdu` | ▪  (TX)<br>▪  (RX) | Bus ECU | ▪ Bus Frame Access<br>▪ Bus PDU Cyclic Timing Control Access<br>▪ Bus PDU Enable Access<br>▪ Bus PDU Length Access<br>▪ Bus PDU Raw Data Access<br>▪ Bus PDU RX Interrupt Access<br>▪ Bus PDU RX Status Access<br>▪ Bus PDU Trigger Access<br>▪ Bus PDU User Code Access<br>▪ Bus ISignal Group<br>▪ Bus ISignal |
| Bus Extended Multiplexed IPDU | `/BusExtendedMultiplexedIPdu` | ▪  (TX)<br>▪  (RX) | Bus ECU | ▪ Bus Frame Access<br>▪ Bus PDU Cyclic Timing Control Access<br>▪ Bus PDU Enable Access<br>▪ Bus PDU Length Access<br>▪ Bus PDU Raw Data Access<br>▪ Bus PDU RX Interrupt Access<br>▪ Bus PDU RX Status Access<br>▪ Bus PDU Trigger Access<br>▪ Bus PDU User Code Access<br>▪ Bus ISignal |
| Bus General-Purpose IPDU | `/BusGeneralPurposeIPdu` | ▪  (TX)<br>▪  (RX) | Bus ECU | ▪ Bus Frame Access<br>▪ Bus PDU Cyclic Timing Control Access<br>▪ Bus PDU Enable Access<br>▪ Bus PDU Length Access<br>▪ Bus PDU Raw Data Access<br>▪ Bus PDU RX Interrupt Access<br>▪ Bus PDU RX Status Access<br>▪ Bus PDU Trigger Access<br>▪ Bus PDU User Code Access |
| Bus General-Purpose PDU | `/BusGeneralPurposePdu` | ▪  (TX)<br>▪  (RX) | Bus ECU | ▪ Bus Frame Access<br>▪ Bus PDU Cyclic Timing Control Access<br>▪ Bus PDU Enable Access<br>▪ Bus PDU Length Access<br>▪ Bus PDU Raw Data Access<br>▪ Bus PDU RX Interrupt Access<br>▪ Bus PDU RX Status Access<br>▪ Bus PDU Trigger Access<br>▪ Bus PDU User Code Access<br>▪ Bus Global Time Domain |
| Bus DCM IPDU | `/BusDcmIPdu` | ▪  (TX) | Bus ECU | ▪ Bus Frame Access |

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| | | ▪  (RX) | | ▪ Bus PDU Cyclic Timing Control Access<br>▪ Bus PDU Enable Access<br>▪ Bus PDU Length Access<br>▪ Bus PDU Raw Data Access<br>▪ Bus PDU RX Interrupt Access<br>▪ Bus PDU RX Status Access<br>▪ Bus PDU Trigger Access<br>▪ Bus PDU User Code Access |
| Bus NMPDU | `/BusNmPdu` | ▪  (TX)<br>▪  (RX) | Bus ECU | ▪ Bus Frame Access<br>▪ Bus PDU Cyclic Timing Control Access<br>▪ Bus PDU Enable Access<br>▪ Bus PDU Length Access<br>▪ Bus PDU Raw Data Access<br>▪ Bus PDU RX Interrupt Access<br>▪ Bus PDU RX Status Access<br>▪ Bus PDU Trigger Access<br>▪ Bus PDU User Code Access<br>▪ Bus ISignal Group<br>▪ Bus ISignal |
| Bus NPDU | `/BusNPdu` | ▪  (TX)<br>▪  (RX) | Bus ECU | ▪ Bus Frame Access<br>▪ Bus PDU Cyclic Timing Control Access<br>▪ Bus PDU Enable Access<br>▪ Bus PDU Length Access<br>▪ Bus PDU Raw Data Access<br>▪ Bus PDU RX Interrupt Access<br>▪ Bus PDU RX Status Access<br>▪ Bus PDU Trigger Access<br>▪ Bus PDU User Code Access |
| Bus User-Defined IPDU | `/BusUserDefinedIPdu` | ▪  (TX)<br>▪  (RX) | Bus ECU | ▪ Bus Frame Access<br>▪ Bus PDU Cyclic Timing Control Access<br>▪ Bus PDU Enable Access<br>▪ Bus PDU Length Access<br>▪ Bus PDU Raw Data Access<br>▪ Bus PDU RX Interrupt Access<br>▪ Bus PDU RX Status Access<br>▪ Bus PDU Trigger Access<br>▪ Bus PDU User Code Access |
| Bus User-Defined PDU | `/BusUserDefinedPdu` | ▪  (TX)<br>▪  (RX) | Bus ECU | ▪ Bus Frame Access<br>▪ Bus PDU Cyclic Timing Control Access<br>▪ Bus PDU Enable Access<br>▪ Bus PDU Length Access<br>▪ Bus PDU Raw Data Access |

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| | | | | ▪ Bus PDU RX Interrupt Access<br>▪ Bus PDU RX Status Access<br>▪ Bus PDU Trigger Access<br>▪ Bus PDU User Code Access |
| Bus ISignal Group | `/BusISignalGroup` | ▪  (TX)<br>▪  (RX) | ▪ Bus ISignal IPDU<br>▪ Bus NMPDU | ▪ Bus ISignal Group End to End Protection Status Access<br>▪ Bus ISignal |
| Bus ISignal | `/BusISignal` | ▪  (TX)<br>▪  (RX) | ▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus NMPDU<br>▪ Bus ISignal Group | ▪ Bus Counter Signal Access<br>▪ Bus ISignal Value Access |
| Bus Global Time Domain | `/BusGlobalTimeDomain` | ▪  (TX)<br>▪  (RX) | Bus General-Purpose PDU | ▪ Bus GTS Time Base Data Access<br>▪ Bus GTS Transmission Control Access<br>▪ Bus GTS Validation Access |
| Bus Communication Controller Enable Access | `/BusCommunicationControllerEnableAccess` |  | ▪ Bus CAN Communication Controller<br>▪ Bus LIN Communication Controller | Function Port |
| Bus Communication Controller LIN Schedule Table Access | `/BusCommunicationControllerLinScheduleTableAccess` |  | Bus LIN Communication Controller | Function Port |
| Bus Communication Controller LIN Wake-Up Access | `/BusCommunicationControllerLinWakeUpAccess` |  | Bus LIN Communication Controller | Function Port |
| Bus J1939 Network Management Enable Access | `/BusJ1939NetworkManagementEnableAccess` |  | Bus CAN Communication Controller | - |
| Bus Frame Access | `/BusFrameAccess` |  | ▪ Bus Container IPDU<br>▪ Bus Multiplexed IPDU<br>▪ Bus Secured IPDU<br>▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus General-Purpose PDU<br>▪ Bus DCM IPDU<br>▪ Bus NMPDU<br>▪ Bus NPDU<br>▪ Bus User-Defined IPDU | Function Port |

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| | | | ▪ Bus User-Defined PDU | |
| Bus PDU Cyclic Timing Control Access | `/BusPduCyclicTimingControlAccess` |  | ▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus General-Purpose PDU<br>▪ Bus DCM IPDU<br>▪ Bus NMPDU<br>▪ Bus NPDU<br>▪ Bus User-Defined IPDU<br>▪ Bus User-Defined PDU | Function Port |
| Bus PDU Enable Access | `/BusPduEnableAccess` |  | ▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus General-Purpose PDU<br>▪ Bus DCM IPDU<br>▪ Bus NMPDU<br>▪ Bus NPDU<br>▪ Bus User-Defined IPDU<br>▪ Bus User-Defined PDU | Function Port |
| Bus PDU Length Access | `/BusPduLengthAccess` |  | ▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus General-Purpose PDU<br>▪ Bus DCM IPDU<br>▪ Bus NMPDU<br>▪ Bus NPDU<br>▪ Bus User-Defined IPDU<br>▪ Bus User-Defined PDU | Function Port |
| Bus PDU Raw Data Access | `/BusPduRawDataAccess` |  | ▪ Bus Container IPDU<br>▪ Bus Multiplexed IPDU<br>▪ Bus Secured IPDU<br>▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus General-Purpose PDU<br>▪ Bus DCM IPDU<br>▪ Bus NMPDU<br>▪ Bus NPDU<br>▪ Bus User-Defined IPDU<br>▪ Bus User-Defined PDU | Function Port |

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| Bus PDU RX Interrupt Access | `/BusPduRxInterruptAccess` |  | ▪ Bus Container IPDU<br>▪ Bus Multiplexed IPDU<br>▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus General-Purpose PDU<br>▪ Bus DCM IPDU<br>▪ Bus NMPDU<br>▪ Bus NPDU<br>▪ Bus User-Defined IPDU<br>▪ Bus User-Defined PDU | ▪ Function Port<br>▪ Event Port |
| Bus PDU RX Status Access | `/BusPduRxStatusAccess` |  | ▪ Bus Container IPDU<br>▪ Bus Multiplexed IPDU<br>▪ Bus Secured IPDU<br>▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus General-Purpose PDU<br>▪ Bus DCM IPDU<br>▪ Bus NMPDU<br>▪ Bus NPDU<br>▪ Bus User-Defined IPDU<br>▪ Bus User-Defined PDU | Function Port |
| Bus PDU SecOC Access | `/BusPduSecOCAccess` |  | Bus Secured IPDU | Function Port |
| Bus PDU Trigger Access | `/BusPduTriggerAccess` |  | ▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus General-Purpose PDU<br>▪ Bus DCM IPDU<br>▪ Bus NMPDU<br>▪ Bus NPDU<br>▪ Bus User-Defined IPDU<br>▪ Bus User-Defined PDU | Function Port |
| Bus PDU User Code Access | `/BusPduUserCodeAccess` |  | ▪ Bus Secured IPDU<br>▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus General-Purpose PDU<br>▪ Bus DCM IPDU | Function Port |

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| | | | ▪ Bus NMPDU<br>▪ Bus NPDU<br>▪ Bus User-Defined IPDU<br>▪ Bus User-Defined PDU | |
| Bus ISignal Group End to End Protection Status Access | `/BusISignalGroupEndToEndProtectionStatusAccess` |  | Bus ISignal Group | Function Port |
| Bus ISignal Value Access | `/BusISignalValueAccess` |  | Bus ISignal | Function Port |
| Bus Counter Signal Access | `/BusCounterSignalAccess` |  | Bus ISignal | Function Port |
| Bus GTS Transmission Control Access | `/BusGtsTransmissionControlAccess` |  | Bus Global Time Domain | Function Port |
| Bus GTS Time Base Data Access | `/BusGtsTimeBaseDataAccess` |  | Bus Global Time Domain | Function Port |
| Bus GTS Validation Access | `/BusGtsValidationAccess` |  | Bus Global Time Domain | Function Port |
| Function Port | `/FunctionPort` | ▪  (function inport)<br>▪  (function outport) | ▪ Bus Configuration Enable Global<br>▪ Bus Communication Controller Enable Access<br>▪ Bus Communication Controller LIN Schedule Table Access<br>▪ Bus Communication Controller LIN Wake-Up Access<br>▪ Bus Frame Access<br>▪ Bus PDU Cyclic Timing Control Access<br>▪ Bus PDU Enable Access<br>▪ Bus PDU Length Access<br>▪ Bus PDU Raw Data Access<br>▪ Bus PDU RX Interrupt Access<br>▪ Bus PDU RX Status Access<br>▪ Bus PDU SecOC Access<br>▪ Bus PDU Trigger Access<br>▪ Bus PDU User Code Access<br>▪ Bus ISignal Group End to End Protection Status Access<br>▪ Bus Counter Signal Access<br>▪ Bus ISignal Value Access | - |

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| | | | ▪ Bus GTS Time Base Data Access<br>▪ Bus GTS Transmission Control Access<br>▪ Bus GTS Validation Access | |
| Event Port | `/EventPort` |  | Bus PDU RX Interrupt Access | - |

**Accessing the Inspection bus configuration part** The following table provides an overview of the XPath expressions for accessing a bus configuration and the elements that can be available for its Inspection part.

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| Bus Configuration | `/BusConfiguration` |  | - | ▪ Bus Configuration Enable Global<br>▪ Bus Configuration Part Inspection |
| Bus Configuration Enable Global | `/BusConfigurationEnableGlobal` |  | Bus Configuration | Function Port |
| Bus Configuration Part Inspection | `/BusConfigurationPartInspection` |  | Bus Configuration | ▪ Bus Communication Matrix<br>▪ Bus Frame Capture |
| Bus Communication Matrix | `/BusCommunicationMatrix` |  | Bus Configuration Part Inspection | ▪ Bus CAN Communication Cluster<br>▪ Bus LIN Communication Cluster |
| Bus Frame Capture | `/BusFrameCapture` |  | Bus Configuration Part Inspection | ▪ Bus Frame Capture Data Inspection<br>▪ Bus Frame Capture Filter |
| Bus CAN Communication Cluster | `/BusCanCommunicationCluster` |  | Bus Communication Matrix | Bus CAN Physical Channel |
| Bus LIN Communication Cluster | `/BusLinCommunicationCluster` |  | Bus Communication Matrix | Bus LIN Physical Channel |
| Bus Frame Capture Data Inspection | `/BusFrameCaptureDataInspection` |  | Bus Frame Capture | Function Port |
| Bus Frame Capture Filter | `/BusFrameCaptureFilter` |  | Bus Frame Capture | ▪ Bus Frame Capture Filter Control Inspection<br>▪ Bus CAN Filter Rule |
| Bus CAN Physical Channel | `/BusCanPhysicalChannel` |  | Bus CAN Communication Cluster | ▪ Bus Container IPDU<br>▪ Bus Multiplexed IPDU<br>▪ Bus Secured IPDU<br>▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus General-Purpose IPDU |

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| | | | | ▪ Bus General-Purpose PDU<br>▪ Bus DCM IPDU<br>▪ Bus NMPDU<br>▪ Bus NPDU<br>▪ Bus User-Defined IPDU<br>▪ Bus User-Defined PDU |
| Bus LIN Physical Channel | `/BusLinPhysicalChannel` |  | Bus LIN Communication Cluster | ▪ Bus ISignal IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus General-Purpose PDU<br>▪ Bus DCM IPDU<br>▪ Bus NMPDU<br>▪ Bus NPDU<br>▪ Bus User-Defined IPDU<br>▪ Bus User-Defined PDU |
| Bus Frame Capture Filter Control Inspection | `/BusFrameCaptureFilterControlInspection` |  | Bus Frame Capture Filter | Function Port |
| Bus CAN Filter Rule | `/BusCanFilterRule` |  | Bus Frame Capture Filter | - |
| Bus Container IPDU | `/BusContainerIPdu` |  (RX) | Bus CAN Physical Channel | ▪ Bus PDU Raw Data Inspection<br>▪ Bus PDU RX Status Inspection |
| Bus Multiplexed IPDU | `/BusMultiplexedIPdu` |  (RX) | Bus CAN Physical Channel | ▪ Bus PDU Raw Data Inspection<br>▪ Bus PDU RX Status Inspection |
| Bus Secured IPDU | `/BusSecuredIPdu` |  (RX) | Bus CAN Physical Channel | ▪ Bus PDU Raw Data Inspection<br>▪ Bus PDU RX Status Inspection<br>▪ Bus PDU SecOC Inspection<br>▪ Bus PDU User Code Inspection |
| Bus ISignal IPDU | `/BusISignalIPdu` |  (RX) | ▪ Bus CAN Physical Channel<br>▪ Bus LIN Physical Channel | ▪ Bus PDU Raw Data Inspection<br>▪ Bus PDU RX Status Inspection<br>▪ Bus PDU User Code Inspection<br>▪ Bus ISignal Group<br>▪ Bus ISignal |
| Bus Extended Multiplexed IPDU | `/BusExtendedMultiplexedIPdu` |  (RX) | Bus CAN Physical Channel | ▪ Bus PDU Raw Data Inspection<br>▪ Bus PDU RX Status Inspection |

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| | | | | ▪ Bus PDU User Code Inspection<br>▪ Bus ISignal |
| Bus General-Purpose IPDU | `/BusGeneralPurposeIPdu` |  (RX) | ▪ Bus CAN Physical Channel<br>▪ Bus LIN Physical Channel | ▪ Bus PDU Raw Data Inspection<br>▪ Bus PDU RX Status Inspection<br>▪ Bus PDU User Code Inspection |
| Bus General-Purpose PDU | `/BusGeneralPurposePdu` |  (RX) | ▪ Bus CAN Physical Channel<br>▪ Bus LIN Physical Channel | ▪ Bus PDU Raw Data Inspection<br>▪ Bus PDU RX Status Inspection<br>▪ Bus PDU User Code Inspection |
| Bus DCM IPDU | `/BusDcmIPdu` |  (RX) | ▪ Bus CAN Physical Channel<br>▪ Bus LIN Physical Channel | ▪ Bus PDU Raw Data Inspection<br>▪ Bus PDU RX Status Inspection<br>▪ Bus PDU User Code Inspection |
| Bus NMPDU | `/BusNmPdu` |  (RX) | ▪ Bus CAN Physical Channel<br>▪ Bus LIN Physical Channel | ▪ Bus PDU Raw Data Inspection<br>▪ Bus PDU RX Status Inspection<br>▪ Bus PDU User Code Inspection<br>▪ Bus ISignal Group<br>▪ Bus ISignal |
| Bus NPDU | `/BusNPdu` |  (RX) | ▪ Bus CAN Physical Channel<br>▪ Bus LIN Physical Channel | ▪ Bus PDU Raw Data Inspection<br>▪ Bus PDU RX Status Inspection<br>▪ Bus PDU User Code Inspection |
| Bus User-Defined IPDU | `/BusUserDefinedIPdu` |  (RX) | ▪ Bus CAN Physical Channel<br>▪ Bus LIN Physical Channel | ▪ Bus PDU Raw Data Inspection<br>▪ Bus PDU RX Status Inspection<br>▪ Bus PDU User Code Inspection |
| Bus User-Defined PDU | `/BusUserDefinedPdu` |  (RX) | ▪ Bus CAN Physical Channel<br>▪ Bus LIN Physical Channel | ▪ Bus PDU Raw Data Inspection<br>▪ Bus PDU RX Status Inspection<br>▪ Bus PDU User Code Inspection |
| Bus ISignal Group | `/BusISignalGroup` |  (RX) | ▪ Bus ISignal IPDU<br>▪ Bus NMPDU | ▪ Bus ISignal Group End to End Protection Status Inspection<br>▪ Bus ISignal |

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| Bus ISignal | `/BusISignal` |  (RX) | ▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus NMPDU<br>▪ Bus ISignal Group | Bus ISignal Value Inspection |
| Bus PDU Raw Data Inspection | `/BusPduRawDataInspection` |  | ▪ Bus Container IPDU<br>▪ Bus Multiplexed IPDU<br>▪ Bus Secured IPDU<br>▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus General-Purpose PDU<br>▪ Bus DCM IPDU<br>▪ Bus NMPDU<br>▪ Bus NPDU<br>▪ Bus User-Defined IPDU<br>▪ Bus User-Defined PDU | Function Port |
| Bus PDU RX Status Inspection | `/BusPduRxStatusInspection` |  | ▪ Bus Container IPDU<br>▪ Bus Multiplexed IPDU<br>▪ Bus Secured IPDU<br>▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus General-Purpose PDU<br>▪ Bus DCM IPDU<br>▪ Bus NMPDU<br>▪ Bus NPDU<br>▪ Bus User-Defined IPDU<br>▪ Bus User-Defined PDU | Function Port |
| Bus PDU SecOC Inspection | `/BusPduSecOCInspection` |  | Bus Secured IPDU | Function Port |
| Bus PDU User Code Inspection | `/BusPduUserCodeInspection` |  | ▪ Bus Secured IPDU<br>▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus General-Purpose PDU<br>▪ Bus DCM IPDU<br>▪ Bus NMPDU<br>▪ Bus NPDU | Function Port |

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| | | | ▪ Bus User-Defined IPDU<br>▪ Bus User-Defined PDU | |
| Bus ISignal Group End to End Protection Status Inspection | `/BusISignalGroupEndToEndProtectionStatusInspection` |  | Bus ISignal Group | Function Port |
| Bus ISignal Value Inspection | `/BusISignalValueInspection` |  | Bus ISignal | Function Port |
| Function Port | `/FunctionPort` | ▪  (function inport)<br>▪  (function outport) | ▪ Bus Configuration Enable Global<br>▪ Bus Frame Capture Data Inspection<br>▪ Bus Frame Capture Filter Control Inspection<br>▪ Bus PDU Raw Data Inspection<br>▪ Bus PDU RX Status Inspection<br>▪ Bus PDU SecOC Inspection<br>▪ Bus PDU User Code Inspection<br>▪ Bus ISignal Group End to End Protection Status Inspection<br>▪ Bus ISignal Value Inspection | - |

**Accessing the Manipulation bus configuration part** The following table provides an overview of the XPath expressions for accessing a bus configuration and the elements that can be available for its Manipulation part.

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| Bus Configuration | `/BusConfiguration` |  | - | ▪ Bus Configuration Enable Global<br>▪ Bus Configuration Part Manipulation |
| Bus Configuration Enable Global | `/BusConfigurationEnableGlobal` |  | Bus Configuration | Function Port |
| Bus Configuration Part Manipulation | `/BusConfigurationPartManipulation` |  | Bus Configuration | Bus Communication Matrix |
| Bus Communication Matrix | `/BusCommunicationMatrix` |  | Bus Configuration Part Manipulation | ▪ Bus CAN Communication Cluster<br>▪ Bus LIN Communication Cluster |

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| Bus CAN Communication Cluster | `/BusCanCommunicationCluster` |  | Bus Communication Matrix | Bus CAN Physical Channel |
| Bus LIN Communication Cluster | `/BusLinCommunicationCluster` |  | Bus Communication Matrix | Bus LIN Physical Channel |
| Bus CAN Physical Channel | `/BusCanPhysicalChannel` |  | Bus CAN Communication Cluster | ▪ Bus Container IPDU<br>▪ Bus Multiplexed IPDU<br>▪ Bus Secured IPDU<br>▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus General-Purpose PDU<br>▪ Bus DCM IPDU<br>▪ Bus NMPDU<br>▪ Bus NPDU<br>▪ Bus User-Defined IPDU<br>▪ Bus User-Defined PDU |
| Bus LIN Physical Channel | `/BusLinPhysicalChannel` |  | Bus LIN Communication Cluster | ▪ Bus ISignal IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus General-Purpose PDU<br>▪ Bus DCM IPDU<br>▪ Bus NMPDU<br>▪ Bus NPDU<br>▪ Bus User-Defined IPDU<br>▪ Bus User-Defined PDU |
| Bus Container IPDU | `/BusContainerIPdu` |  (TX) | Bus CAN Physical Channel | ▪ Bus Frame Length Manipulation<br>▪ Bus Suspend Frame Transmission Manipulation |
| Bus Multiplexed IPDU | `/BusMultiplexedIPdu` |  (TX) | Bus CAN Physical Channel | ▪ Bus Frame Length Manipulation<br>▪ Bus Suspend Frame Transmission Manipulation |
| Bus Secured IPDU | `/BusSecuredIPdu` |  (TX) | Bus CAN Physical Channel | ▪ Bus Frame Length Manipulation<br>▪ Bus Suspend Frame Transmission Manipulation<br>▪ Bus PDU SecOC Authenticator Invalidation Manipulation<br>▪ Bus PDU SecOC Freshness Overwrite Value Manipulation<br>▪ Bus PDU User Code Manipulation |
| Bus ISignal IPDU | `/BusISignalIPdu` |  (TX) | ▪ Bus CAN Physical Channel<br>▪ Bus LIN Physical Channel | ▪ Bus Frame Length Manipulation<br>▪ Bus Suspend Frame Transmission Manipulation |

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| | | | | ▪ Bus PDU User Code Manipulation<br>▪ Bus ISignal Group<br>▪ Bus ISignal |
| Bus Extended Multiplexed IPDU | `/BusExtendedMultiplexedIPdu` |  (TX) | Bus CAN Physical Channel | ▪ Bus Frame Length Manipulation<br>▪ Bus Suspend Frame Transmission Manipulation<br>▪ Bus PDU User Code Manipulation<br>▪ Bus ISignal |
| Bus General-Purpose IPDU | `/BusGeneralPurposeIPdu` |  (TX) | ▪ Bus CAN Physical Channel<br>▪ Bus LIN Physical Channel | ▪ Bus Frame Length Manipulation<br>▪ Bus Suspend Frame Transmission Manipulation<br>▪ Bus PDU User Code Manipulation |
| Bus General-Purpose PDU | `/BusGeneralPurposePdu` |  (TX) | ▪ Bus CAN Physical Channel<br>▪ Bus LIN Physical Channel | ▪ Bus Frame Length Manipulation<br>▪ Bus Suspend Frame Transmission Manipulation<br>▪ Bus PDU User Code Manipulation |
| Bus DCM IPDU | `/BusDcmIPdu` |  (TX) | ▪ Bus CAN Physical Channel<br>▪ Bus LIN Physical Channel | ▪ Bus Frame Length Manipulation<br>▪ Bus Suspend Frame Transmission Manipulation<br>▪ Bus PDU User Code Manipulation |
| Bus NMPDU | `/BusNmPdu` |  (TX) | ▪ Bus CAN Physical Channel<br>▪ Bus LIN Physical Channel | ▪ Bus Frame Length Manipulation<br>▪ Bus Suspend Frame Transmission Manipulation<br>▪ Bus PDU User Code Manipulation<br>▪ Bus ISignal Group<br>▪ Bus ISignal |
| Bus NPDU | `/BusNPdu` |  (TX) | ▪ Bus CAN Physical Channel<br>▪ Bus LIN Physical Channel | ▪ Bus Frame Length Manipulation<br>▪ Bus Suspend Frame Transmission Manipulation<br>▪ Bus PDU User Code Manipulation |
| Bus User-Defined IPDU | `/BusUserDefinedIPdu` |  (TX) | ▪ Bus CAN Physical Channel<br>▪ Bus LIN Physical Channel | ▪ Bus Frame Length Manipulation<br>▪ Bus Suspend Frame Transmission Manipulation<br>▪ Bus PDU User Code Manipulation |
| Bus User-Defined PDU | `/BusUserDefinedPdu` |  (TX) | ▪ Bus CAN Physical Channel | ▪ Bus Frame Length Manipulation |

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| | | | ▪ Bus LIN Physical Channel | ▪ Bus Suspend Frame Transmission Manipulation<br>▪ Bus PDU User Code Manipulation |
| Bus ISignal Group | `/BusISignalGroup` |  (TX) | ▪ Bus ISignal IPDU<br>▪ Bus NMPDU | Bus ISignal |
| Bus ISignal | `/BusISignal` |  (TX) | ▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus NMPDU<br>▪ Bus ISignal Group | ▪ Bus Feature Switch<br>▪ Bus ISignal Offset Value Manipulation<br>▪ Bus ISignal Overwrite Value Manipulation |
| Bus Feature Switch | `/BusFeatureSwitch` |  | Bus ISignal | Function Port |
| Bus Frame Length Manipulation | `/BusFrameLengthManipulation` |  | ▪ Bus Container IPDU<br>▪ Bus Multiplexed IPDU<br>▪ Bus Secured IPDU<br>▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus General-Purpose PDU<br>▪ Bus DCM IPDU<br>▪ Bus NMPDU<br>▪ Bus NPDU<br>▪ Bus User-Defined IPDU<br>▪ Bus User-Defined PDU | Function Port |
| Bus Suspend Frame Transmission Manipulation | `/BusSuspendFrameTransmissionManipulation` |  | ▪ Bus Container IPDU<br>▪ Bus Multiplexed IPDU<br>▪ Bus Secured IPDU<br>▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus General-Purpose PDU<br>▪ Bus DCM IPDU<br>▪ Bus NMPDU<br>▪ Bus NPDU<br>▪ Bus User-Defined IPDU<br>▪ Bus User-Defined PDU | Function Port |
| Bus PDU SecOC Authenticator Invalidation Manipulation | `/BusPduSecOCAuthenticatorInvalidationManipulation` |  | Bus Secured IPDU | Function Port |
| Bus PDU SecOC Freshness Overwrite Value Manipulation | `/BusPduSecOCFreshnessOverwriteValueManipulation` |  | Bus Secured IPDU | Function Port |

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| Bus PDU User Code Manipulation | `/BusPduUserCodeManipulation` |  | ▪ Bus Secured IPDU<br>▪ Bus ISignal IPDU<br>▪ Bus Extended Multiplexed IPDU<br>▪ Bus General-Purpose IPDU<br>▪ Bus General-Purpose PDU<br>▪ Bus DCM IPDU<br>▪ Bus NMPDU<br>▪ Bus NPDU<br>▪ Bus User-Defined IPDU<br>▪ Bus User-Defined PDU | Function Port |
| Bus ISignal Offset Value Manipulation | `/BusISignalOffsetValueManipulation` |  | Bus ISignal | Function Port |
| Bus ISignal Overwrite Value Manipulation | `/BusISignalOverwriteValueManipulation` |  | Bus ISignal | Function Port |
| Function Port | `/FunctionPort` | ▪  (function inport)<br>▪  (function outport) | ▪ Bus Configuration Enable Global<br>▪ Bus Frame Length Manipulation<br>▪ Bus PDU SecOC Authenticator Invalidation Manipulation<br>▪ Bus PDU SecOC Freshness Overwrite Value Manipulation<br>▪ Bus PDU User Code Manipulation<br>▪ Bus Feature Switch<br>▪ Bus ISignal Offset Value Manipulation<br>▪ Bus ISignal Overwrite Value Manipulation | - |

**Accessing a bus configuration and its Gateways part** The following table provides an overview of the XPath expressions for accessing a bus configuration and its Gateways part.

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| Bus Configuration | `/BusConfiguration` |  | - | ▪ Bus Configuration Enable Global<br>▪ Bus Configuration Part Gateways |
| Bus Configuration Enable Global | `/BusConfigurationEnableGlobal` |  | Bus Configuration | Function Port |
| Bus Configuration Part Gateways | `/BusConfigurationPartGateways` |  | Bus Configuration | Bus Frame Gateway |

| Element Type | XPath Expression | Symbol | Parent Elements | Child Elements |
|---|---|---|---|---|
| Bus Frame Gateway | `/BusFrameGateway` |  | Bus Configuration Part Gateways | Bus Frame Gateway Direction |
| Bus Frame Gateway Direction | `/BusFrameGatewayDirection` |  | Bus Frame Gateway | Function Port |
| Bus Frame Gateway Filter | `/BusFrameGatewayFilter` |  | Bus Frame Gateway | ▪ Bus Filter Control Gateways<br>▪ Bus CAN Filter Rule |
| Bus Filter Control Gateways | `/BusFilterControlGateways` |  | Bus Frame Gateway Filter | Function Port |
| Bus CAN Filter Rule | `/BusCanFilterRule` |  | Bus Frame Gateway Filter | - |
| Function Port | `/FunctionPort` |  (function inport) | ▪ Bus Configuration Enable Global<br>▪ Bus Frame Gateway Direction<br>▪ Bus Filter Control Gateways | - |

**Examples**

The following listing provides short examples for accessing specific elements of a bus configuration.

```
# Accessing all BusConfigurations
'/BusConfiguration'

# Accessing all BusCommunicationMatrices in all BusConfigurations
'/BusConfiguration//BusCommunicationMatrix'

# Accessing all BusMultiplexedIPdus in all BusEcus whose name contains 'door' in all BusCommunicationMatrices in
# all BusConfigurationPartSimulatedEcus in all BusConfigurations
'/BusConfiguration/BusConfigurationPartSimulatedEcus/BusCommunicationMatrix/BusEcu[contains(@Name,
"door")]/BusMultiplexedIPdu'

# Accessing all BusContainerIPdus in all BusEcus in all BusCommunicationMatrices in
# all BusConfigurationPartSimulatedEcus in all BusConfigurations
'/BusConfiguration/BusConfigurationPartSimulatedEcus/BusCommunicationMatrix/BusEcu/BusContainerIPdu'

# Accessing all FunctionPorts with IsMappable = 'False' of all BusPduRawDataAccess features
# in 'Bus Configuration (1)'
'/BusConfiguration[@Name = "Bus Configuration (1)"]//BusPduRawDataAccess/FunctionPort[@IsMappable = "False"]'

# Accessing all FunctionPorts with IsTestAutomationSupportEnabled = 'True' of the BusISignalValueAccess Feature in
# the BusConfigurationPartInspection of 'Bus Configuration (1)'
'/BusConfiguration[@Name = "Bus Configuration
(1)"]/BusConfigurationPartInspection//BusISignalValueAccess/FunctionPort[@IsTestAutomationSupportEnabled = "True"]'
```

# Accessing Element Properties via XPath

**Introduction**

ConfigurationDesk's automation interface provides special relations for accessing the properties of communication matrix elements and bus configuration elements via XPath expressions.

**Relations for accessing element properties via XPath**

Many elements of communication matrices and bus configurations provide properties which you can access via XPath expressions. To access the properties via XPath, you must use one of the following relations:

- `CommunicationMatricesByClustersWithProperties`
- `CommunicationMatricesByEcusWithProperties`
- `BusConfigurationsWithProperties`

These relations are elements of the ICaRelations <<Collection>> on page 160 interface.

**Note**

The relations with properties decrease the performance. Access as few properties as possible via these relations.

To access elements without their properties, use the `CommunicationMatricesByClusters`, `CommunicationMatricesByEcus`, or `BusConfigurations` relations instead.

If you have to access a high number of element properties (e.g., all the properties of all the PDUs of a communication matrix) use the `PropertyDataObjects` relation in combination with the `CommunicationMatricesByClusters`, `CommunicationMatricesByEcus`, or `BusConfigurations` relations. In this case, you cannot access the properties via XPath expressions.

**XPath expressions**

To access elements of communication matrices and bus configurations, you can use the same XPath expressions as for the `CommunicationMatricesByClusters`, `CommunicationMatricesByEcus`, and `BusConfigurations` relations.

To access the element properties, you can use the XPath expressions that are listed in the following table. Use these expressions as lower-level elements of the elements whose properties you want to access. For examples on the syntax, refer to Examples on page 82. The slash (`/`) is part of the XPath expression.

| XPath Expression | Element Symbol in Properties Browser | Available For | Description |
|---|---|---|---|
| `/BusCanCommunicationConnector` |  | ▪ `BusNetworkNode` | Accesses a CAN connector specified for a cluster. |

| XPath Expression | Element Symbol in Properties Browser | Available For | Description |
|---|---|---|---|
| | | ▪ BusEcu (only for the CommunicationMatricesByEcusWithProperties relation) | |
| /BusLinCommunicationConnector |  | ▪ BusNetworkNode<br>▪ BusEcu (only for the CommunicationMatricesByEcusWithProperties relation) | Accesses a LIN connector specified for a cluster. |
| /BusJ1939TransportProtocolNode |  | ▪ BusNetworkNode<br>▪ BusEcu (only for the CommunicationMatricesByEcusWithProperties relation) | Accesses a J1939 transport protocol node (J1939 TP node). |
| /BusJ1939NmNode |  | ▪ BusNetworkNode<br>▪ BusEcu (only for the CommunicationMatricesByEcusWithProperties relation) | Accesses a J1939 network management node (J1939 NM node). |
| /BusFrame |  | ▪ BusISignalIPDU<br>▪ BusGeneralPurposeIPdu<br>▪ BusGeneralPurposePdu<br>▪ BusDcmIPdu<br>▪ BusNmPdu<br>▪ BusNPdu<br>▪ BusUserDefinedIPdu<br>▪ BusUserDefinedPdu<br>▪ BusMultiplexedIPdu<br>▪ BusContainerIPdu | Accesses a frame of a PDU. |
| /BusLinScheduleTableEntry |  | ▪ BusISignalIPDU<br>▪ BusGeneralPurposeIPdu<br>▪ BusGeneralPurposePdu<br>▪ BusDcmIPdu<br>▪ BusNmPdu<br>▪ BusNPdu<br>▪ BusUserDefinedIPdu<br>▪ BusUserDefinedPdu | Accesses the entries of a LIN schedule table. |
| /BusJ1939TransportProtocolConnection |  | ▪ BusISignalIPDU<br>▪ BusMultiplexedIPdu | Accesses a J1939 transport protocol connection. |
| /BusCyclicTiming |  | BusISignalIPDU | Accesses a Cyclic timing node of a PDU. |
| /BusEventControlledTiming |  | BusISignalIPDU | Accesses an Event-controlled timing node of a PDU. |
| /BusTimeRange | – | BusISignalIPDU | Accesses the time range of the related higher-level element. |
| /BusDynamicPart |  | BusMultiplexedIPdu | Accesses the dynamic part of a multiplexed IPDU. |
| /BusStaticPart |  | BusMultiplexedIPdu | Accesses the static part of a multiplexed IPDU. |

| XPath Expression | Element Symbol in Properties Browser | Available For | Description |
|---|---|---|---|
| `/BusSegmentPosition` |  | `BusMultiplexedIPdu` | Accesses the segment position of the dynamic or static part of a multiplexed IPDU. |
| `/BusSelectorField` |  | `BusMultiplexedIPdu` | Accesses the selector field of a multiplexed IPDU. |
| `/BusDynamicPartAlternative` |  | `BusMultiplexedIPdu` (only for the `BusConfigurationsWithProperties` relation) | Accesses the different basic PDUs that are defined for a multiplexed IPDU. |
| `/BusEndToEndDescription` |  | `BusISignalGroup` | Accesses the end-to-end protection definition of an ISignal group. |
| `/BusEndToEndProtectionISignalIPdu` |  | `BusEndToEndDescription` | Accesses the position of an end-to-end protected ISignal group in a PDU. |
| `/BusEndToEndTransformer` |  | `BusISignalGroup` | Accesses the end-to-end transformer configuration of an ISignal group. |
| `/BusISignalToIPduMapping` |  | `BusISignal` | Accesses properties concerning the mapping of an ISignal to a PDU. |
| `/BusCodedType` |  | `BusISignal` | Accesses the coded data type of an ISignal. |
| `/BusPhysicalType` |  | `BusISignal` | Accesses the physical data type of an ISignal. |
| `/BusComputationMethod` |  | `BusISignal` | Accesses a computation method of an ISignal. |
| `/BusComputationScale` |  | `BusISignal` | Accesses a computation scale defined for a computation method of an ISignal. |
| `/BusTimeMaster` |  | `BusGlobalTimeDomain` | Accesses a time master of a global time domain. |
| `/BusTimeSlave` |  | `BusGlobalTimeDomain` | Accesses a time slave of a global time domain. |

**Examples**

The following listing provides short examples for accessing properties of communication matrix elements sorted by clusters.

```
# CommunicationMatricesByClustersWithProperties

# Accessing the names of all BusCanCommunicationConnectors in all BusNetworkNodes whose name contains 'door' in
# all BusCanCommunicationClusters in all CAN BusSystems in all BusCommunicationMatrices
'/BusCommunicationMatrix/BusSystemCan/BusCanCommunicationCluster/BusNetworkNode[contains(@Name,
"door")]/BusCanCommunicationConnector/@Name'

# Accessing all BusSelectorFields whose name contains 'MuxPDU' in all BusMultiplexedIPdus in
# all BusCanPhysicalChannels
'//BusCanPhysicalChannel/BusMultiplexedIPdu/BusSelectorField[contains(@Name, "MuxPDU")]'
```

The following listing provides short examples for accessing properties of communication matrix elements sorted by ECUs.

```
# CommunicationMatricesByEcusWithProperties

# Accessing all Identifiers of all BusFrames in all BusCanPhysicalChannels in all BusCanCommunicationClusters in
# all BusISignalIPdus with the 'TX' direction of the 'DoorLeft' BusEcu in all BusCommunicationMatrices
'/BusCommunicationMatrix/BusEcu[@Name = "LeftDoor"]/BusISignalIPdu[@Direction =
"TX"]/BusCanCommunicationCluster/BusCanPhysicalChannel/BusFrame/@Identifier'

# Accessing all BusISignals whose name contains 'Signal' and whose lower-level elements are
# BusPhysicalType with Base_data_type 'UINT8'
'//BusPhysicalType[@Base_data_type = "UINT8"]/parent::BusISignal[contains(@Name, "Signal")]'
```

The following listing provides short examples for accessing properties of bus configurations elements.

```
# BusConfigurationsWithProperties

# Accessing the BusCodedType of all BusISignals with the 'TX' direction and regardless of whether they are
# included in a BusISignalGroup. The BusISignals are accessed in any PDU type of all BusEcus in
# any BusCommunicationMatrix that are assigned to the Simulated ECUs part of any BusConfiguration
'/BusConfiguration/BusConfigurationPartSimulatedEcus/BusCommunicationMatrix/BusEcu/*/descendant-or-
self::BusISignal[@Direction="TX"]/BusCodedType'
```

# Examples of Automating Bus Manager Features

### Configuring a bus configuration and generating bus configuration structures

The following listing shows a basic workflow for configuring a bus configuration:

1. Import a communication matrix.
2. Add a bus configuration to the active ConfigurationDesk application.
3. Assign communication matrix elements to different parts of the bus configuration.
4. Add bus configuration features to elements of the bus configuration.
5. Enable model access for function ports.
6. Generate the model interface to be used in a MATLAB/Simulink behavior model and in ConfigurationDesk.

The listing is only a short example and does not show all the possible tasks of working with bus configurations. It is assumed that a project and an application are still opened in ConfigurationDesk.

**Tip**

The listing uses the *chassis_changed.dbc* communication matrix, which is available with the *CANMMDemo* demo project. If you want to run the script, you have to adjust only the path of the communication matrix.

```
# Initialize certain basic variables
CurrentApplication = Application.ActiveApplication
BusManager = CurrentApplication.Components.Item('BusManager')
```

```
BusConfigurationsRelation = CurrentApplication.Relations.Item('BusConfigurations')
BusConfigurationsWithPropertiesRelation = CurrentApplication.Relations.Item('BusConfigurationsWithProperties')
BusConfigurationType = BusConfigurationsRelation.GetCreatableTypes().Item(0)
CommunicationMatricesByEcusRelation = CurrentApplication.Relations.Item('CommunicationMatricesByEcus')

# Add a communication matrix to the active ConfigurationDesk application
CommunicationMatrixFile = r'C:\Users\<current user>\Documents\dSPACE\ConfigurationDesk\<current version>\CANMMDemo\CfgCANMMDemo\SLModel\chassis_changed.dbc'
BusManager.Configure('AddCommunicationMatrix', [CommunicationMatrixFile])
CommunicationMatrix = CommunicationMatricesByEcusRelation.GetTopNodes().Item('chassis_changed')

# Add a new bus configuration to the active ConfigurationDesk application
BusConfiguration = BusConfigurationsRelation.CreateDataObject(BusConfigurationType)
BusConfiguration.Name = 'New Bus Configuration'

# Assign the 'IGNITION' ECU to the Simulated ECUs bus configuration part of the 'New Bus Configuration'
EcuIgnition = CommunicationMatricesByEcusRelation.GetElements(CommunicationMatrix).Item('IGNITION')
BusManager.Configure('AssignElements', [[EcuIgnition], BusConfiguration])

# Remove the 'Diagnostics' IPDUs from the 'New Bus Configuration'
DiagnosticsIPdus = BusConfigurationsRelation.FindByXPath('/BusConfiguration[@Name = "New Bus Configuration"]//BusMultiplexedIPdu[contains(@Name, "DIAGNOSTICS")]|/BusConfiguration[@Name = "New Bus Configuration"]//BusDynamicPartIPdu[contains(@Name, "DIAGNOSTICS")]')
BusManager.Configure('RemoveElements', [DiagnosticsIPdus])

# Select the 'IGNITION_1' IPDU in the Simulated ECUs bus configuration part and add the PDU Trigger feature to it
IPDUIgnition1BusCfgPartSimulation = BusConfigurationsRelation.FindByXPath('/BusConfiguration[@Name = "New Bus Configuration"]/BusConfigurationPartSimulatedEcus//BusISignalIPdu[contains(@Name, "IGNITION_1") and @Direction = "TX"]')
BusManager.Configure('AddFeature', ['BusPduTriggerAccess', IPDUIgnition1BusCfgPartSimulation[0]])

# Access the Inspection part of the bus configuration, select the 'IGNITION_1' IPDU in
# the communication matrix, and assign the IPDU to the Inspection part
Inspection = BusConfigurationsRelation.GetElements(BusConfiguration).Item('Inspection')
IPDUIgnition1ComMatrix = CommunicationMatricesByEcusRelation.GetElements(EcuIgnition).Item('IGNITION_1')
BusManager.Configure('AssignElements', [[IPDUIgnition1ComMatrix], Inspection])

# Add the ISignal Value feature to all ISignals of the 'IGNITION_1' IPDU that are assigned to the Inspection part
ISignalsIgnition1BusCfgPartInspection = BusConfigurationsRelation.FindByXPath('/BusConfiguration[@Name = "New Bus Configuration"]/BusConfigurationPartInspection//BusISignalIPdu[contains(@Name, "IGNITION_1") and @Direction = "RX"]/BusISignal')
for ISignal in ISignalsIgnition1BusCfgPartInspection:
   BusManager.Configure('AddFeature', ['BusISignalValueInspection', ISignal])

# Enable 'Model access' for all function ports
for FunctionPortModelAccess in BusConfigurationsRelation.FindByXPath('/BusConfiguration[@Name = "New Bus Configuration"]//FunctionPort/@IsMappable'):
   FunctionPortModelAccess.TrySetValue(True)

# Generate the model interface for the bus configuration function ports with enabled model access in
# ConfigurationDesk and a new Simulink model
CurrentApplication.Algorithms.PropagateToSimulink([BusConfiguration])
```

### Modifying communication matrices and accessing the modified elements

The following listing shows:

- How to add elements to communication matrices and assign the added elements to a bus configuration.
- How to specify user-defined settings for communication matrix elements.

- How to access the modified communication matrix elements.
- How to write an overview of the modified elements to ConfigurationDesk's Interpreter.

The listing provides only a short example and does not show all the possible aspects of modifying and accessing communication matrix elements. It is assumed that a project and an application are still open in ConfigurationDesk and a communication matrix with at least one physical CAN channel is available in the ConfigurationDesk application.

```
# Initializing certain basic variables
CurrentApplication = Application.ActiveApplication
BusManager = CurrentApplication.Components.Item('BusManager')
BusConfigurationsRelation = CurrentApplication.Relations.Item('BusConfigurations')
BusConfigurationsWithPropertiesRelation = CurrentApplication.Relations.Item('BusConfigurationsWithProperties')
BusConfigurationType = BusConfigurationsRelation.GetCreatableTypes().Item(0)
CommunicationMatricesByClustersRelation = CurrentApplication.Relations.Item('CommunicationMatricesByClusters')

# Select the 'CanBodyPhysicalChannel' CAN channel of the 'BusManagerDemo' communication matrix and add
# one ISignal IPDU and two ISignals to the channel
CanPhysicalChannel = CommunicationMatricesByClustersRelation.FindByXPath('/BusCommunicationMatrix[@Name =
"BusManagerDemo"]//BusCanPhysicalChannel[@Name = "CanBodyPhysicalChannel"]')[0]
BusManager.Configure('AddElementToCommunicationMatrix', ['BusISignalIPdu',
CanPhysicalChannel, 'BusDirectedElementTX'])
NewBusISignalIPdu = CommunicationMatricesByClustersRelation.FindByXPath('/BusCommunicationMatrix[@Name =
"BusManagerDemo"]//BusCanPhysicalChannel[@Name = "CanBodyPhysicalChannel"]/BusISignalIPdu[@Name =
"User_defined_IPDU_1"]')[0]
for i in range(2):
    BusManager.Configure('AddElementToCommunicationMatrix', ['BusISignal', NewBusISignalIPdu])

# Assign the user-defined IPDU to 'Bus Configuration (1)'
BusConfiguration1 = BusConfigurationsRelation.FindByXPath('/BusConfiguration[@Name = "Bus Configuration (1)"]')
if len(BusConfiguration1) == 0:
    BusConfiguration1 = BusConfigurationsRelation.CreateDataObject(BusConfigurationType)
else:
    BusConfiguration1 = BusConfiguration1[0]
BusManager.Configure('AssignElements', [[NewBusISignalIPdu], BusConfiguration1])

# Change the length of the user-defined IPDU
Result = NewBusISignalIPdu.Properties['Length'].TrySetValue(24)
print('Changed length of user-defined IPDU from "{0!s}" to "24" -> {1!s}\n'.format(NewBusISignalIPdu.Name, Result))

# Change the length of 'User_defined_ISignal_1'
NewISignal = BusConfigurationsRelation.FindByXPath('//BusISignal[@Name = "User_defined_ISignal_1"]')[0]
Result = NewISignal.Properties['Length'].TrySetValue(16)
print('Changed length of user-defined ISignal from "{0!s}" to "16" -> {1!s}\n'.format(NewISignal.Name, Result))

# Change the coded and physical base data types of 'User_defined_ISignal_1' to UINT16
ISignalCodedType = BusConfigurationsWithPropertiesRelation.FindByXPath('//BusISignal[@Name =
"User_defined_ISignal_1"]/BusCodedType')[0]
ISignalPhysicalType = BusConfigurationsWithPropertiesRelation.FindByXPath('//BusISignal[@Name =
"User_defined_ISignal_1"]/BusPhysicalType')[0]
# Base data types:
# '1': 'INT8'
# '2': 'UINT8'
# '3': 'INT16'
# '4': 'UINT16'
# '5': 'INT32'
# '6': 'UINT32'
# '7': 'INT64'
```

```
# '8': 'UINT64'
# '9': 'FLOAT32'
# '10': 'FLOAT64'
# '17': 'BOOLEAN'
ElementName = ISignalCodedType.Name
Result = ISignalCodedType.Properties['Base data type'].TrySetValue('4')
print('Changed base data type of "{0!s}" to "UINT16" -> {1!s}\n'.format(ElementName, Result))
ElementName = ISignalPhysicalType.Name
Result = ISignalPhysicalType.Properties['Base data type'].TrySetValue('4')
print('Changed base data type of "{0!s}" to "UINT16" -> {1!s}\n'.format(ElementName, Result))

# Select the modified communication matrix elements that are assigned to 'Bus Configuration (1)'
ModifiedElements = BusConfigurationsWithPropertiesRelation.FindByXPath('/BusConfiguration[@Name = "Bus Configuration
(1)"]//*[@Changes_to_communication_matrix = "True"]')

# Continue only if 'ModifiedElements' is not empty
if len(ModifiedElements) == 0:
    print('No modified communication matrix element found!')
else:
    # Print a list of the modified elements to ConfigurationDesk's Interpreter
    print('\nModified communication matrix elements:\n')
    for ModifiedElement in ModifiedElements:
       for Property in ModifiedElement.properties:
          print('{0!s}: {1!s}'.format(Property.name, Property.value))
       print('\n')
```

# Best Practices for Automating ConfigurationDesk

**Objective**

There are some tips and tricks that will make your work with the ConfigurationDesk automation API easier.

**Where to go from here**

Information in this section



## Best Practices for Using External Interpreters

**Running scripts in PythonWin**

Running scripts in PythonWin might take considerably longer than in other external interpreters, such as Python.exe. Use PythonWin only to use its debugging features. Refer to How to Use PythonWin's Debugger on page 28.

## Best Practices for Script Optimization

**Avoid to iterate collections**

Iterating collections is time-consuming, especially with process boundaries. You should therefore try to avoid iterating a collection more often than necessary.

For example, there are different ways to set a subset of property values of a collection:

**Example of slow approach with many iterations**

```
if MyDeviceBlock.Properties.Contains("PhysicalAttributes"):
   pPhysicalAtt = MyDeviceBlock.Properties.Item("PhysicalAttributes")
if MyDeviceBlock.Properties.Contains("Name"):
   pName = MyDeviceBlock.Properties.Item("Name")
if MyDeviceBlock.Properties.Contains("PortType"):
   pPortType = MyDeviceBlock.Properties.Item("PortType")
```

Depending on the position of the properties `PhysicalAttribute`, `Name`, and `PortType` in the device blocks properties collection, and assuming that no

caching is implemented, the implementation can iterate up to six times through almost the whole collection.

**Example of fast approach avoiding many iterations (recommended)**

```
properties = block.Properties
ValueDict = {}
for p in properties:
Name = p.Name
if Name == "PhysicalAttributes":
   ValueDict["PhysicalAttributes "] = p.Value
if Name == "Name":
   ValueDict["Name"] = p.Value
if Name == "PortType":
   ValueDict["PortType"] = p.Value
if len(ValueDict) == 3:
   break
```

This implementation iterates through the collection only once, therefore saving time.

## Use automation transactions

Transactions are useful to encapsulate and separate functionality. With the automation interface, you can create autonomous read or write transactions that are closed after performing their automation task.

Transactions offer the following benefits:

- A read transaction ensures that no changes are made to the data model while data is being read.
- A write transaction ensures that no other client can change the data in the meantime and that the changes can be undone in one step.
- Transactions speed up the execution because of they block GUI updates during processing.

**Example of a transaction enabling test automation** The following script represents a function `SetTestAutomationActive` which iterates through the function library (`FuncLib`) of an active application (`activeApp`) to activate test automation for all function instances in the application.

```
WriteTransaction = None
try:
   WriteTransaction =
activeApp.TransactionCreator.CreateWriteTransaction("TAAutomation")
   FuncTypes = FuncLib.Item(0)
   for FuncType in FuncTypes:
      for Func in FuncType:
         SetTestAutomationActive(Func)
except:
   print("Exception")
finally:
   WriteTransaction.Close()
```

**Note**

There are two important constraints for using transactions:

- Use an automation transaction only in an external interpreter to avoid blocking.
- Enclose the transaction in a `try` - `finally` statement to ensure that the transaction will be closed even if an exception was raised.

**Use ICaAlgorithms**

The ICaAlgorithms interface provides some methods to perform a configuration task for one or more ICaDataObjects. For example, it is possible to create a suitable model port block for every instantiated function block. If you need to perform recurrent tasks, consider using ICaAlgorithms.

**Example of slow approach**

```
# do something, then call
[…]
activeApp.Algorithms.CreateSuitableModelPortBlock([FunctionInstance])
[…]
# then do something and call
[…]
activeApp.Algorithms.CreateSuitableModelPortBlock([FunctionInstance])
[…]
# again do something and call
[…]
activeApp.Algorithms.CreateSuitableModelPortBlock([FunctionInstance])
```

With a small number of function instances, there is no performance issue, but calling this script for many function instances may cause considerable overhead for inter-process communication.

**Example of fast approach**

```
# collect function instances and call method once
[…]
activeApp.Algorithms.CreateSuitableModelPortBlock(MyFunctionInstances)
```

# Best Practices for Interfaces

**Non-typed return values in C#**

In some cases the return value of a ConfigurationDesk automation API method is a collection of non-typed objects. In C# it may be necessary to check the type of the object to access it correctly.

With the following example script, you can get a group of different objects (working views, blocks, links) that you have selected in ConfigurationDesk.

```
ICaObjects obs = ActiveApplication.GetSelectedObjects("", "");
```

```
foreach (Object obj in obs)
{
   ICaWorkingView wv = obj as ICaWorkingView;
   if (wv != null)
      Console.WriteLine("WorkingView: " + wv.FullName);
```

```
   ICaDataObject db = obj as ICaDataObject;
   if (db != null)
      Console.WriteLine("DataObject: " + db.Name);
```

```
    ICaLink lk = obj as ICaLink;
    if (lk != null)
      Console.WriteLine("Link" + lk.ImplementingType);
}
```

## Best Practices for Data Structures and Parameters

### Notes on using default parameters

The automation interface has some methods that use default parameters. The different programming languages affect these methods in different ways.

**Python** In Python, there are no issues with using default parameters in methods of the automation interface. For example, there is a default String-type parameter called "Name" for creating a child object using the ICaDataObject interface. A simple call is possible:

```
NewChild = MyParent.CreateChild(MyParent.DataObjectTypes.Item(0))
```

**C#** Even though C# in general supports default parameters, it is not possible to use them in ConfigurationDesk automation. The interface definitions of the automation API use the DefaultParameterValue attribute to mark a parameter as default. This declaration forbids the use of a default value for C# but is necessary to enable default parameters for scripting clients like Python and VB. Thus, the call of the CreateChild method for C# looks as follows:

```
NewChild = MyParent.CreateChild(MyParent.DataObjectTypes.Item(0),String.Empty);
```

**MATLAB** In MATLAB M-code it is not possible to use default parameters for ConfigurationDesk automation at all. Use quotes for empty strings as defaults:

```
NewChild = MyParent.CreateChild(MyParent.DataObjectTypes.Item(int32(0)),'');
```

Or use square brackets for an empty array:

```
Relation.AddElements(Parent, {Element}, []);
```

### Notes on untyped arrays as parameters

In some cases, for example, in the ICaAlgorithms interface, the ConfigurationDesk automation API uses methods with untyped array parameters:

```
void CreateSuitableModelPortBlock([MarshalAs(UnmanagedType.SafeArray,
SafeArraySubType =
System.Runtime.InteropServices.VarEnum.VT_VARIANT)] Array FunctionBlocks);
```

There are different approaches for this in C#, Python and MATLAB:

**C#** Methods with untyped array parameters should be called with an array list:

```
System.Collections.ArrayList arrList = new System.Collections.ArrayList();
arrList.Add(MyFunctionBlock);
ActiveApplication.Algorithms.CreateSuitableModelPortBlock(arrList.ToArray());
```

**Python** In Python you can use a list:

```
ActiveApplication.Algorithms.CreateSuitableModelPortBlock([MyFunctionBlock]);
```

**MATLAB** If you need an array, you must use a MATLAB-specific cell array which is stated in braces:

```
ActiveApplication.Algorithms.CreateSuitableModelPortBlock({MyFunctionBlock});
```

For an empty array, use brackets:

```
Relation.AddElements(Parent, {Element}, []);
```

### Notes on string arrays as parameters

In some cases, for example, in the ICaAlgorithms interface, the ConfigurationDesk automation API uses methods with string arrays as parameters:

```
void ExportTableViewToXML(String FullPath, String TableName,
String[] Infos,
[DefaultParameterValue("")] String ColumnsSet);
```

While this signature causes no trouble in C# or Python, you will get problems in MATLAB if you want to provide an empty string array.

**MATLAB** The problem is to convert an empty safe array from BSTR to a string array, which can be successfully done only with one single value. For this reason, a special feature should be activated in MATLAB before calling the automation method:

```
feature('COM_SafeArraySingleDim', 1)
Algorithms.ExportTableViewToXML('c:\MyTest.xml', 'Function
Electrical Interface', {''},'');
```

If you want to provide one or more string values:

```
Algorithms.ExportTableViewToXML('c:\test.xml', 'Function Electrical
Interface', {'OneString' },'');
Algorithms.ExportTableViewToXML('c:\test.xml', 'Function Electrical
Interface', {'OneString'; 'AnotherString'},'');
```

### Notes on vectors as property values

Some properties expect vectors to set their value and return a vector representing this value. For example, an Injection/Ignition Current In function block provides access to a property **Number of expected pulses** (automation name: NumberOfExpectedPulses) that is of type vector and whose length depends on the **Number of event windows** property (automation name: NumberOfEventWindows). This example is approached as follows in C#, Python and MATLAB:

**C#** Use an ArrayList to get the vector of the property value and to access an item of it:

```
ArrayList Arr = NumbPulsProp.Value as ArrayList;
int Val = (int) Arr[0];
```

Setting the value means setting a complete vector, for example:

```
Arr[0] = 3;
NumbPulsProp.Value = Arr;
```

**Python** Use the value property of the automation object to get this item in Python:

```
Vals = NumbPulsProp.Value
Val = Vals[0]
```

Setting the value is similar to C#:

```
Vals[0] = 3
NumbPulsProp.Value = Vals
```

**MATLAB** MATLAB M-scripts do not support direct access to such COM objects. Therefore, you have to invoke the item method with an appropriate index:

```
NumbPulsProp =
InjectionIgnition.Properties.Item('NumberOfExpectedPulses');
Val = invoke(NumbPulsProp.Value, 'Item', int32(0));
```

Changing a specific value of the vector:

```
Val = NumbPulsProp.Value;
invoke(Val, ‘RemoveAt’, int32(0));
invoke(Val, ‘Insert’, int32(0), 3);
NumbPulsProp.Value = Val;
```

Setting a complete vector is more convenient:

```
NumbPulsProp.Value = [3; 2];
```

### Creating types

Creating types via the ICaDataObject method CreateChild needs an ICaDataObjectType which can be queried from the parent object. Always use the correct ICaDataObject to query the type and then create the child object.

Not recommended:

```
deviceporttype = MyDevice.DataObjectTypes.Item("DevicePort")
MyOtherDevice.CreateChild(deviceporttype)
```

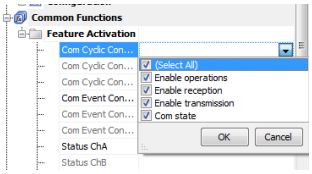
Instead, use:

```
deviceporttype = MyOtherDevice.DataObjectTypes.Item("DevicePort")
MyOtherDevice.CreateChild(deviceporttype)
```

### Using lists for sets of values

If you need a set of values for a property, use a list to set them properly.

For example, the properties of the Common Functions item in a FlexRay function block are represented in the Properties Browser in a list with checkboxes, as illustrated below.




For automation, use a list as follows:

```
propertylist = ["Enable operations", "Enable reception"]
prop.Value = propertylist
```

To print the values, iterate the list:

```
for value in prop.Value:
   print(f)
```

```
Enable operations
Enable reception
Enable transmission
Com state
```

## Best Practices for Property Handling

**Using the DisplayName property**

As of dSPACE Release 2015-B, the ICaProperty interface contains the DisplayName property. You can use it to find the automation name of a property displayed in the **Properties Browser**. For example, a **Voltage In** function block has a **Capture angle position** property, whose name is different from its automation name: IsAnglePositionEnabled. The reason is that a display name can change in the future while an automation name must not. Therefore, avoid using display names in an automation script.

**Setting a property value**

You can set a property value either directly by assigning it to the ICaProperty.Value property or indirectly by using the ICaProperty.TrySetValue(<Object>) method, which returns true or false indicating if the assignment was successful. For notes on specific data structures and parameters, see Best Practices for Data Structures and Parameters on page 90.

# Basics on Python Relevant for Automation

**Where to go from here**

Information in this section

# Basics on Python

**Where to go from here**

Information in this section



## Main Characteristics of Python

**Introduction**

"Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for rapid application development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse." *("Python Reference Manual")*

To get an idea of the Python programming language, you need to know its main characteristics.

**Variable type definition**

No explicit variable type definition is necessary. For example, you do not have to declare a variable as an integer or float variable before assigning the values, just assign the values:

```
a = 25
b = 1.23
```

**Scopes and indentation**

Control structures – for example loops, if constructs, functions, class definitions – do not use an end command. The scopes are declared by indentation, for example:

```
if a > 10:
   print('outer if')
if b > 50:
   print ('inner if')
else:
   print ('inner else')
else:
   print ('outer else')
```

For an example, see lesson 4 (refer to Python Demo Scripts on page 102).

**Comments**

Comments begin with a "#" and extend to the end of the line.

**Print function**

Any variable can be displayed using the print command. It has formatting parameters similar to the C language printf function.

**String representation**

If you need a string representation of any variable or expression, use the built-in function `str`: see lesson 1 (refer to Python Demo Scripts on page 102).

**Module import**

The scope of any variable can be controlled by different methods. There are local and global symbols. Symbols from other modules can be imported using different methods:

- Qualification by dot notation:

```
# access the os module and
# use the method splitext of the os.path module
import os
(File, Extension) = os.path.splitext('MyFile.txt')
```

- Qualified import of individual symbols of another module:

```
# global import and
# direct use of method splitext
from os.path import splitext
(FileName, Extension) = splitext('MyFile.txt')
```

The most suitable method depends on the specific situation. A symbol taken from a qualified or global import can then no longer be used by dot notation. It overwrites an existing symbol with the same name regardless of its type.

**Data structures**

Python has a rich set of predefined data structures, such as lists, tuples, dictionaries and corresponding operators and methods. Refer to lesson 2 (refer to Python Demo Scripts on page 102) and lesson 6 (refer to Python Demo Scripts on page 102).

**Functions**

Functions can be defined at any location in a script. There are several ways to call functions and define default values for function parameters. Parameter passing

can be performed by position or by name. Refer to lesson 7 (refer to Python Demo Scripts on page 102).

**File operations**

File operations provide read and write access to files managed by the Windows operating system. Basic operations are opening and closing a file with different access modes and positioning the file pointer. Refer to lesson 9 (refer to Python Demo Scripts on page 102).

**Objects**

Python offers object-oriented description features, such as class definitions and multiple inheritance: see lesson 10 (refer to Python Demo Scripts on page 102).

**Error handling**

Python uses exception handling as error handling. Both user-defined and standard exceptions can be used: see lesson 8 (refer to Python Demo Scripts on page 102).

**PYC files**

Reusable script modules can be compiled into an internal byte representation for improved performance. These files have the PYC file name extension.

**Tip**

- You will find the corresponding example scripts in the `.\Demos\Python\Tutorial` folder of your ConfigurationDesk installation.
- In the examples, the screen output produced by the Python Interpreter is integrated into the script. The output is printed in bold type and preceded by ">>>" so that you can recognize it easily.

**Related topics**

Basics

Basics on Python ..... 96

Examples

Python Demo Scripts ..... 102

# Multithreaded Scripting

**Introduction**

To execute automation scripts in parallel, you can run each script in a separate thread.

You can execute an automation script

- In the main thread or
- In a separate thread (multithreading).

**Main thread**

If you execute a script in the main thread, all other ConfigurationDesk activities are blocked. You cannot execute another script or any other action in parallel.

**Separate thread (multithreading)**

If you run an automation script in a separate thread, more than one script can be executed in parallel. Typical reasons for multithreaded script execution are:

- The program's screen has to be updated during script execution.
- You want to change parameters in the executing program.
- The script has to wait for a certain amount of time or an event without blocking the program during script execution.

**Note**

Only the main thread should be used for developing scripts.

**Tip**

You might consider using an external Python interpreter (see Using an External Interpreter on page 26) in preference to multithreaded scripting.

**Executing a thread**

Multithreaded execution is achieved in different ways:

- Using the standard Python `_thread` module. For example:

```
import _thread
def f(Param):
   for i in range(Param):
      print(i)
_thread.start_new_thread(f, (1000,))
```

- Using the standard Python `threading` module. For example:

```
import threading
def f(Param):
   for i in range(Param):
      print(i)
t = threading.Thread(target=f, args=(1000,))
t.start()
```

**Note**

It is recommended to use the threading module only if you need to program threads directly, because it allows you to have more control and information.

### Stopping a thread

A thread is stopped when

- The script or the function ends (preferred way) or
- The script or the function is interrupted via the context menu of the program's Interpreter symbol in the system tray:  .

You can also stop a thread by another thread by checking and changing a variable which is accessible in both threads. This could be a global variable, a variable in a common module, etc. From time to time, the other thread has to check this variable for the break condition. If a break has been set, the thread can finish normally and clean up the objects.

For an example, see lesson 11 (refer to Python Demo Scripts on page 102).

### Multithreading rules

For multithreading, you have to observe the following rules:

- If you use Python objects which represent COM objects, for example, created by `win32com.client.Dispatch`:
  - Always add a call to `CoInitialize` and `CoUninitialize` when using threads, for example:

```
def ThreadFunction():
  import pythoncom
  try:
    pythoncom.CoInitialize()
    ComObject = win32.client.Dispatch(...)
  finally:
    ComObject = None
    pythoncom.CoUninitialize()
```

  - Do not pass these objects between threads as parameters.
  - Do not use these objects with the `global` statement.
- Polling loops within a separate thread must have a call to `Sleep` from either the *win32api* or the *time* Python module. This ensures that the separate thread does not block other threads.
- If you run a script in a separate thread, it is not allowed to create dialogs. For example, you cannot use wxwindows or raw_input.

**Tip**

To stop the execution of a separate thread, you can use the MessageBox provided by the *win32api* Python module.

### Related topics

Examples

# Python Code Examples

**Where to go from here**

Information in this section



# Examples of Python Scripts

**Introduction**

The basic tasks and operations are explained by short Python examples. This includes object-oriented features and exception handling.

**Where to go from here**

Information in this section



Information in other sections

# Python Demo Scripts

**Location of the demo scripts**

You can find Python demo scripts in the following subfolders of your dSPACE RCP & HIL installation folder:

- `.\Demos\Python\Tutorial`
- `.\Demos\Python\ProgrammingMultithreadedApplications`

**Overview of example scripts**

The following example scripts are available:

| Script file | Description |
|---|---|
| `lesson_01_StringHandling.py` | Demonstrates basic and advanced features of string handling and the converting of strings to other types. |
| `lesson_02_ListHandling.py` | Demonstrates how to define lists and use their predefined operators and methods.<br>You can think of a list object as an array of Python objects. The content of a list can be modified. Lists can be resized and sorted. Lists are heterogeneous, that is, there is no restriction on the type of objects that can be contained in a list, and objects of different types can reside in the same list. |
| `lesson_03_SetHandling.py` | Demonstrates basic features of sets. |
| `lesson_04_ControlStructures.py` | Demonstrates the definition of control structures. |
| `lesson_05_SequenceHandling.py` | Demonstrates features of general sequence handling (e.g. tuple and list). |
| `lesson_06_DictionaryHandling.py` | Demonstrates how to define dictionaries and use their predefined operators and methods.<br>Python dictionaries associate constant Python objects (the keys) with other arbitrary Python objects (the values). For example, you can associate strings to phone numbers (represented as integers). |
| `lesson_07_ArgumentHandling.py` | Demonstrates how to use functions and handle arguments.<br>Functions can be defined at any location in a script. If the script has been executed once, the function can be reused without reloading the script.<br>Default values for function arguments can be defined. Argument passing can be performed by position or by name.<br>Arguments can be derived from any Python type. The type can be changed at run time, that is, functions can be overloaded. |
| `lesson_08_ExceptionHandling.py` | Demonstrates how to define exceptions. There is an extensive set of predefined exceptions which can be used by exception handlers. User-defined exceptions can also be used. The "except" part can be made conditionally, which is not shown in this example.<br>For further information on error handling, refer to *Built-in Exceptions* in the *Python Library Reference*. |
| `lesson_09_FileHandling.py` | Demonstrates basic file I/O operations, such as opening and closing a file with different access modes, and writing, reading and positioning the file pointer.<br>For further information on file operations, refer to *File Objects* in the *Python Library Reference*. |
| `lesson_10_ObjectOriented.py` | Demonstrates several object-oriented features of Python, in particular the inheritance of functions.<br>For further information on object-oriented features, refer to *Data Model* and *Compound Statements* in the *Python Language Reference*. |

| Script file | Description |
|---|---|
| `lesson_11_ThreadingBasics.py` | Demonstrates how to stop a thread from another thread using the threading module to start the threads. |
| `lesson_12_ThreadingWaiting.py` | Demonstrates how to synchronize different threads of execution by using a lock object. |
| `lesson_13_ThreadingWaitingAdvanced.py` | Demonstrates how to wait in the main thread for other threads which are executed. |
| `lesson_14_UseThreadingBaseClass.py` | Demonstrates how to use the dSPACE threading base class. |

**Related topics**

Basics

Main Characteristics of Python ..... 96

# Examples of Translating Python Code into Different Programming Languages

**Objective**

The following examples show the same working steps in languages: Python, Visual Basic (VB) , MATLAB M-file (M), and C#.

**Where to go from here**

Information in this section

Control Structures ..... 104
Short examples of working with control structures show the main differences between different programming languages.

Line Continuation ..... 105
Short examples of working with line continuation show the main differences between different programming languages.

Creation ..... 105
Short examples of creation show the main differences between different programming languages.

Destruction ..... 106
Short examples of destruction show the main differences between different programming languages.

Calling Methods without Parameters ..... 106
Short examples of calling methods without parameters show the main differences between different programming languages.

Information in other sections



## Control Structures

**Python**

```
if Item.EnumProperty == ENUM_VALUE and\
   Item.StringProperty == "String":
      ...
```

**Visual Basic**

```
If Item.EnumProperty = ServerLib.ENUM_VALUE
    If Item.StringProperty = "String" Then
        ...
    End If
```

**M-Code**

```
if Item.EnumProperty == ENUM_VALUE
    if strcmp(Item.StringProperty, 'String')
        ...;
    end;
end;
```

**C#**

```
if  (  item.EnumProperty  == EnumClass.ENUM_VALUE
    && item.StringProperty.Equals("String")
{
    ...
}
```

## Line Continuation

**Python**

```
CallMethodWithParameter(\
Parameter ...
```

**Visual Basic**

```
CallMethodWithParameter ( _
Parameter ...
```

**M-Code**

```
CallMethodWithParameter ( ...
Parameter);
```

**C#**

```
CallMethodWithParameter (
    Parameter);
```

## Creation

**Python**

```
Server = Dispatch('Server.Object.1')
```

**Visual Basic**

```
Set Server = CreateObject("Server.Object.1")
```

**M-Code**

```
Server = actxserver('Server.Object.1');
```

**C#**

```
Type remoteType = Type.GetTypeFromProgID(
   "Server.Object.1", "127.0.0.1", true);
object remoteObj = Activator.CreateInstance(remoteType);
IServerInterface server = remoteObj as IServerInterface;
```

## Destruction

**Python**

```
del Server
```

**Visual Basic**

```
Set Server = Nothing
```

**M-Code**

```
clear Server;
```

**C#**

```
Marshal.ReleaseComObject(server);
this.server = null;
```

## Calling Methods without Parameters

**Python**

With parentheses:

```
Server.CallMethod()
```

**Visual Basic**

Without parentheses:

```
Server.CallMethod
```

**M-Code**

Without parentheses:

```
Server.CallMethod;
```

**C#**

With parentheses:

```
Server.CallMethod ();
```

## Collections

**Python**

Indexing possible:

```
Server.Collection[0]
for Item in Server.Collection:
```

**Visual Basic**

Indexing possible:

```
Server.Collection(0)
For Each Item In Server.Collection
```

**M-Code**

Indexing not possible:

```
Server.Collection.Item(int32(0))
for Index = 0:( Server.Collection.Count - 1)
    Item = Server.Collection.Item(int32(Index));
end;
```

**Note**

By default, MATLAB uses the `double` data type when you call a method with a parameter, for example, like this:

```
Item(0)
```

However, since parameters of the `double` data type cannot be handled by the automation interface of ConfigurationDesk, you have to cast the data type of such a parameter to `Int32` in your M-code.

**C#**

```
Server.Collection[0]
foreach (string Item in Server.Collection)
{
   ...
}
```

## Constants

**Python**

Access via global variables:

```
ENUM_VALUE = 1
```

**Visual Basic**

Access via type information, add reference library to your VB project:

```
ServerLib.ENUM_VALUE
```

**M-Code**

Not accessible

**C#**

```
EnumClass.ENUM_VALUE
```

## Array Handling

**Python**

Simple brackets:

```
Server.ArrayProperty = [0,0]
```

**Visual Basic**

Declaration and setting of each value:

```
Dim Values(2) As Variant
Values(0) = 0
Values(1) = 0
Server.ArrayProperty = Values
```

**M-Code**

Simple brackets:

```
Server.ArrayProperty = [0,0];
```

**C#**

```
object[] values = new object[2] { 0, 0 };
Server.ArrayProperty = values;
```

# Code Examples Showing Programming Constructs

**Objective**

Code examples in different programming languages show how to create an object, access properties and methods, and select items from a collection.

**Note**

The code examples just show the ways the programming constructs should be translated. They do not work on real servers.

**Where to go from here**

Information in this section



# Code Example in Python

**Example**

The following example shows how to create an object, access properties and methods, and select items from a collection.

**Note**

The code examples just show the ways the programming constructs should be translated. They do not work on real servers.

```
from win32com.client import Dispatch
ENUM_VALUE = 1
Server = Dispatch('Server.Object.1')
StringValue = Server.StringProperty
print(StringValue)
Item = Server.Collection[0]
Collection = Server.Collection
for Item in Collection:
    if Item.EnumProperty == ENUM_VALUE and\
       Item.StringProperty == "String":
       Item.CallMethod()
       Item.IntegerProperty = 42
ArrayValue = Server.ArrayProperty
print(ArrayValue)
del StringValue
del ArrayValue
del Item
del Collection
Server.Quit()
del Server
```

**Tip**

You can find more demo scripts written in Python in the `\Demos\ConfigurationDesk\Toolautomation\Python` demo folder.

## Code Example in Visual Basic

**Example**

The following example shows how to create an object, access properties and methods, and select items from a collection.

**Note**

The code examples just show the ways the programming constructs should be translated. They do not work on real servers.

```
...
Dim ArrayValue As Variant
Set Server = CreateObject("Server.Object.1")
StringValue = Server.StringProperty
Debug.Print StringValue
Set Item = Server.Collection(0)
Set Collection = Server.Collection
For Each Item In Collection:
    If Item.EnumProperty = ServerLib.ENUM_VALUE Then
        If Item.StringProperty = "String" Then
            Item.CallMethod
            Item.IntegerProperty = 42
        End If
    End If
Next
ArrayValue = Server.ArrayProperty
For Each ArrayItem In ArrayValue
    Debug.Print ArrayItem
Next
Set StringValue = Nothing
Set ArrayValue = Nothing
Set Item = Nothing
Set Collection = Nothing
Server.Quit
Set Server = Nothing
...
```

## Code Example in M-Code

**Example**

The following example shows how to create an object, access properties and methods, and select items from a collection.

**Note**

The code examples just show the ways the programming constructs should be translated. They do not work on real servers.

```
ENUM_VALUE = 1;
Server = actxserver('Server.Object.1');
StringValue = Server.StringProperty;
disp(sprintf('%s', StringValue));
Item = Server.Collection.Item(int32(0));
Collection = Server.Collection;
for Index = 0:(Collection.Count - 1)
  Item = Collection.Item(int32(Index));
  if Item.EnumProperty == ENUM_VALUE
    if strcmp(Item.StringProperty, 'String')
      Item.CallMethod();
      Item.IntegerProperty = 42;
    End;
  end;
end;
ArrayValue = Server.ArrayProperty;
CellArray = cell2mat(ArrayValue);
for Index = 1:(length(CellArray))
    disp(sprintf('%d', CellArray(Index)));
end;
clear StringValue;
clear CellArray;
clear ArrayValue;
clear Item;
clear Collection;
Server.Quit();
clear Server;
```

## Code Example in C#

**Example**

The following example shows how to create an object, access properties and methods, and select items from a collection.

**Note**

The code examples just show the ways the programming constructs should be translated. They do not work on real servers.

```
using Company.Program.Interfaces;
...
Type serverType = Type.GetTypeFromProgID("Server.Object.1");
IServerInterface server = Activator.CreateInstance(
   serverType) as IServerInterface;
string stringValue = server.StringProperty;
Console.WriteLine(stringValue);
IItemInterface item = server.Collection[0];
ICollectionInterface collection = server.Collection;
foreach (item in collection)
{
    if (   item.EnumProperty  == EnumClass.ENUM_VALUE
        && item.StringProperty.Equals("String"))
    {
        item.CallMethod();
        item.IntegerProperty = 42;
    }
}
object[] arrayValue = (object[])server.ArrayProperty;
for (object arrayItem in arrayValue)
{
    Console.Write(String.Format("{0:d} ", (int)arrayItem));
}
...
```

# Limitations and Important Changes

**Objective**

ConfigurationDesk lets you automate most of its features, but there are a few limitations for its automation interface.

Due to new ConfigurationDesk features or dependencies to other products changes are made to the automation interface for each release.

**Where to go from here**

Information in this section



## Limitations for Automating ConfigurationDesk

**Global limitations**

Do not use ConfigurationDesk's automation interface when a modal ConfigurationDesk dialog is currently open. For example, if you close the project and application via automation and a modal ConfigurationDesk dialog is currently open, this can lead to inconsistencies or even to ConfigurationDesk crashing.

**Limitations concerning the project root folder**

You cannot lock your ConfigurationDesk automation session against other automated ConfigurationDesk accesses, such as automated access via AutomationDesk. All entities that are involved in automation have to cooperate.

There is no mechanism that protects changes or configurations by one entity from those made by others.

**Limitations for automating project and application management**

Some limitations affect the automated handling of project and application management. The limitations shown below are listed in order of the automation objects which are affected.

**General Limitations**

- Using an automation ICaApplication object from the ICaApplications collection of an ICaProject and not from an ICaActiveProject may lead to unpredictable results. To work with applications, activate the project and use the ICaApplications collection of the ICaActiveProject interface. The ICaApplicationMain interface provides quick access to an open project or to an active application via its ICaApplicationMain.ActiveProject and ICaApplicationMain.ActiveApplication properties.

**ICaApplicationMain**

- `OpenProject/OpenApplication`: Using these methods for opening a project requires that the project, project folder and project file have the same name.

  Example:

  - Project name: MyProject
  - Project folder: `<Documents folder>\MyProject`
  - Project file: `MyProject.cdp`

**ICaActiveProject**

- `Close`: Closing a project with write-protected elements

  You get an error message if you close a project with write-protected project elements via automation and the `SaveChanges` parameter of the `Close()` method is set to `TRUE`. As a workaround, call the method with the `SaveChanges` parameter set to `FALSE`: `ActiveProject.Close(False)`. Do not use the `Save()` method for projects with write-protected project elements.
- `SaveAs`: A project can be saved under a new name only in the root folder of the original project. After modifying a project, you must always save it with the `ActiveProject.Save()` method first before you use the `ActiveProject.SaveAs()` method.
- `SaveTo`: Unsaved changes in the active project are not only saved in the target directory, but also in the source root directory.

**ICaApplication**

- `Import` is currently not implemented. Using `Import` will throw an exception.

**ICaApplications**

- `Add/Item`: It is not possible to add two applications whose names differ only in that one of them has the application file name extension (`.cdl`) . For example, the application with the name `Application_001.cdl` will not be found with the item method of the ICaApplications object if there is another application with the name `Application_001` present in the project.

**ICaActiveApplication**

- `Files` is currently not supported. The returned collection will always be empty.
- `Export` is currently not implemented. Using `Export` will throw an exception.
- `SaveAs` is currently not implemented. Using `SaveAs` will throw an exception.
- `Rename` is currently not implemented. Using `Rename` will throw an exception.

**ICaFiles**

- This interface is intended for later use.

## Limitations for automating window handling

Automated window handling is only supported for ConfigurationDesk's main window. If you start ConfigurationDesk via automation, its user interface is not displayed immediately. To display it, you should use the `Visible` property of the `ICaMainWindow` interface.

## Limitations for automating the signal chain

Some signal chain configuration tasks are not available in automation, so MATLAB configuration cannot be automated via ConfigurationDesk.

The following limitations are listed in order of the automation objects which are affected.

**ICaComponent**

- `Configure`: An attempt to create or replace a hardware topology from a hardware system which was registered a short time ago, e.g., in an automation step directly before, might run into problems because of race conditions. See the dSPACE demo scripts for appropriate programming steps to avoid this.
- `Configure`: A property of a communication matrix element can be passed to the `UndoChangesToCommunicationMatrix` operation only via the property's automation name but not via its object itself.

**ICaWorkingViews**

- As of ConfigurationDesk 4.3, it is possible to create working views and working view groups with an empty name or the same name on the same root level. Because the automation interfaces cannot distinguish between working views and/or working view groups with an empty name or the same name on the same level, it is strongly recommended not to use the same or empty names.
- As of ConfigurationDesk 4.3, it is possible to create working views and working view groups with slashes and backslashes in their names. However, you must avoid slashes and backslashes in names because the automation interfaces needs the names of working views and working view groups as path information.

**ICaDataObject**

- `Parent`: Not all data objects which are items in the signal chain have a parent, and even the topology browser shows the items as child nodes.

- `Equals`: It is not possible to compare Bus Manager elements of different relations. For example, you cannot compare Bus Manager elements of the `CommunicationMatricesByEcus` relation with Bus Manager elements of the `CommunicationMatricesByClusters` or `BusConfigurations` relation. You can only compare Bus Manager elements of the same relation.

**ICaProperty**

- The automation interface treats some property values of data objects as strings rather than as other simple types. For example, the `Slots` property of some boards should be simple integer values, but if the values are not read-only you can set them only as strings (e.g.: `'2'` instead of: `2`).
- Some property values are displayed as strings in the Properties Browser, but the underlying type is a vector of integers. The value of such a property can be read as a vector with `Property.Value[0]`. The value itself is set as a string (e.g. `Property.Value = '2'`). (For example, the Injection/Ignition Current In function block type from the function library provides the Pulse Cut State function port with vector values.)

**ICaTransaction**

- Using a write transaction in ConfigurationDesk's internal interpreter might lead to a blocking call. Use write transactions only with external calls from threads other than the UI thread.
- In each write transaction, you can add only one bus configuration to a ConfigurationDesk application. To add multiple bus configurations to a ConfigurationDesk application, you must use a separate write transaction for each bus configuration.

**ICaRelation**

- `FindByXPath`: Currently only supported for communication matrix relations and bus configuration relations.

**TransferType** The TransferType Enumeration member ExpectedLoadDescription is obsolete. The corresponding value now refers to load rejection.

**Limitations for automating function block configuration**

**Assigned master APU provider only assignable via name string** For an Engine Simulation Setup block it is not possible to assign a master APU provider by setting the value to the appropriate object. Use the name string of the object instead, for example:

```
ApuProviderProperty.Value = "Angular Clock Setup (1)"
```

## New Features and Changes to the Automation Interface for Release 2021-A

**Python 3.9**

Due to the end of life of Python 3.6 until end of December 2021, dSPACE decided to switch to Python 3.9 with dSPACE Release 2021-A. For more

information, refer to Migrating Python Scripts from Python 3.6 to Python 3.9 (New Features and Migration 🕮).

For information on how the switch to Python 3.9 affects the Message Reader API, refer to Reading dSPACE Log Messages via the Message Reader API on page 250.

**SPI communication properties**

The following changes to SPI communication properties might lead to existing scripts using the **Time between words** property to not work correctly:

The new **Word separation** property lets you select the method to separate consecutive data words within an SPI cycle. Until Release 2020-A, you select the method by setting the **Time between words** property to 0 s (chip select inactive between words) or to a value greater than 0 s (transmission pause between words).

**ICaComponent BusManager**

**Configure operation AddFrameCapture** You can add a frame capture to the Inspection part of a specific bus configuration.

**Configure operation AddFilterRule** You can add a filter rule to a specific frame capture filter or frame gateway filter.

Refer to ICaComponent <<Collection>> on page 141.

# Version-Specific Migration Steps for ConfigurationDesk Tool Automation

**6.5 to 6.6 (dSPACE Release 2020-B)**

**Changes to SENT In / SENT Out properties** The following changes were applied to properties of the **SENT In** and **SENT Out** function blocks:

- The CRCCalculation property from the **Serial Message** function port group was moved to the function block level and renamed SerialMessageCRCCalculation.
- The CRCCalculation property from the **Protocol** function port group was moved to the function block level and renamed ProtocolCRCCalculation.

**ApplicationProcessToModel enumeration** To avoid using the reserved Python keyword None, the ApplicationProcessToModel enumeration value None was changed to No.

### 6.4 to 6.5 (dSPACE Release 2020-A)

The automation names of the following function block properties were changed:

| Function Block Type | Display Name | Previous Automation Name | New Automation Name |
|---|---|---|---|
| Current In | Trigger source | MeasurementMode | TriggerSource |
| Voltage In | Trigger source | MeasurementMode | TriggerSource |
| PWM/PFM In | Function mode | MeasurementMode | FunctionMode |

**Note**

You must adjust the automation names of these properties in existing scripts for them to work correctly.

### 6.3 to 6.4 (dSPACE Release 2019-B)

**Note**

The following changes affect the data model and can cause code from previous Releases to malfunction.

**ICaProperty** The former FeatureActivator property of an Ethernet Setup function block is substituted by the IsDefaultGatewayEnabled property (display name: 'Default gateway').

**ICa Component: Configure Create-operation** You can specify how preconfigured application processes should be generated if you create a model topology by importing an MCD file.

The support of multiple application processes for MCD files is realized by an additional parameter with the default value True. This can cause the unwanted generation of multiple application processes in existing scripts.

### 6.1 to 6.2 (dSPACE Release 2018-B)

**Discontinuation of Python 2.7 support** The support of Python 2.7 is discontinued with dSPACE Release 2018-B. Python 3.6 is now supported.

**Note**

The following changes affect the data model and can cause code from previous Releases to malfunction.

**ICaRelation/ICaRelations** The following changes affect automation scripts, including the related XPath queries, that automate Bus Manager features and/or access Bus Manager elements.

As of ConfigurationDesk 6.1p1 and Bus Manager 6.1p1, the following changes apply to the `BusMultiplexedIPdu` primary role:

- The `Sequence number` property of the `Selector Field`, `Dynamic Part`, and `Static Part` entities is obsolete and was removed.

- The `Switch Code` property of the `Dynamic Part` entity was renamed to `Selector field code`.

**6.0 to 6.1 (dSPACE Release 2018-A)**

**Note**

The following changes affect the data model and can cause code from previous Releases to malfunction.

**ICaRelation/ICaRelations** The following changes affect automation scripts, including the related XPath queries, that automate Bus Manager features and/or access Bus Manager elements.

The primary role `BusFeature` is divided into the following primary roles:

- `BusConfigurationEnableGlobal`
- `BusCommunicationControllerEnableAccess`
- `BusCommunicationControllerLinScheduleTableAccess`
- `BusCommunicationControllerLinWakeUpAccess`
- `BusFrameAccess`
- `BusPduCyclicTimingControlAccess`
- `BusPduEnableAccess`
- `BusPduRawDataAccess`
- `BusPduTriggerAccess`
- `BusCounterSignalAccess`
- `BusISignalValueAccess`
- `BusPduRawDataInspection`
- `BusPduRxStatusInspection`
- `BusISignalValueInspection`
- `BusSuspendFrameTransmissionManipulation`
- `BusISignalOffsetValueManipulation`
- `BusISignalOverwriteValueManipulation`

**ICaDataObject** The following changes affect automation scripts that access Bus Manager elements.

`Name`: The naming scheme for function ports of bus configurations has changed from `<name of bus configuration element>_<FunctionPortFunction>` to `<name of bus configuration element> <Function Port Function>`. For example, the name of the function port to access a LIN schedule table of a LIN master named MasterNode has changed from `MasterNode_ScheduleIndex` to `MasterNode Schedule Index`.

**5.7 to 6.0 (dSPACE Release 2017-B)**

**Note**

The following changes affect the data model and can cause code from previous releases to malfunction.

**ICaApplicationMain** `SetCustomInformation` / `GetCustomInformation`: The possibility to set or to get the `ExtendSignalChainOptions` has been removed. Use the properties `ModelPortBlockStructure` (grouped, ungrouped) and `ModelPortDataType` (Float64, Inherited) of the relevant function block type to set the appropriate values.

**ICaComponent** The following changes affect automation scripts, including the related XPath queries, that automate Bus Manager features and/or access Bus Manager elements.

- `Configure`: The operations `GenerateStructuredModel` and `UpdateConnectedModelPortBlocks` are obsolete and no longer supported. Instead, use the following operations:
  - Generating a new Simulink model: `ICaAlgorithms::PropagateToSimulink` with `targetModel = <null>`
  - Updating connected model port blocks: `ICaAlgorithms::PropagateToSimulink` with `targetModel = <connected Simulink model>`
- `Configure`: The operations `AddUserDefinedElement` and `UndoUserDefinedChanges` were renamed to `AddElementToCommunicationMatrix` and `UndoChangesToCommunicationMatrix`.

**ICaRelation/ICaRelations** The following changes affect automation scripts, including the related XPath queries, that automate Bus Manager features and/or access Bus Manager elements.

- The role names of some bus configuration features changed:
  - `BusIPduRawDataAccess` changed to `BusPduRawDataAccess`.
  - `BusIPduTriggerAccess` changed to `BusPduTriggerAccess`.
  - `BusIPduEnableAccess` changed to `BusPduEnableAccess`.
- The role names of some Bus Manager elements changed:
  - `BusIPdu` changed to `BusPdu`.
  - `BusCfgElement` changed to `BusConfigurationElement`.
- The primary role `BusPhysicalChannel` is divided into two new primary roles, `BusCanPhysicalChannel` and `BusLinPhysicalChannel`.
- The primary role `BusCommunicationCluster` is divided into two new primary roles, `BusCanCommunicationCluster` and `BusLinCommunicactionCluster`.
- The primary role `BusISignalIPdu` is divided into the following primary roles:
  - `BusISignalIPdu`
  - `BusUserDefinedIPdu`
  - `BusUserDefinedPdu`
  - `BusDcmIPdu`
  - `BusGeneralPurposeIPdu`
  - `BusGeneralPurposePdu`
  - `BusNmPdu`
  - `BusNPdu`

- The `Pdu type` property of the primary PDU roles is obsolete and was removed. Instead, the following new properties are available according to the primary PDU role:
  - `Category` (new `BusPduCategory` secondary role)
  - `Diagnostic PDU type` (new `BusDiagnosticPdu` secondary role)
  - `XCP configuration` (new `BusXcpConfiguration` secondary role)
- The property `Has user-defined changes` is renamed to `Changes to communication matrix`.

**ICaWorkingView** `ShowSignalChain`, `ShowModelCommunication`: The parameter `NewWindow` is no longer supported. It always will be set to false.

**ICaProperty** `EdgeType` / `TriggerCondition`: The property `Trigger` of an I/O trigger block provider is renamed from `EdgeType` to `TriggerCondition` (e.g., Multi-Channel PWM Out).

### 5.6 to 5.7 (dSPACE Release 2017-A)

**Note**

The following changes affect the data model and can cause code from previous releases to malfunction.

**ICaProperty/ICaProperties**

- Due to changes to property categories it is possible that the order of properties of an `ICaDataObject` that you get is different from previous releases.
- The property `MeasurementMode` was shifted from the electrical interface (internal name: signal conditioning) level of a function block to the function block level.
- Changing the property `IsFailureSimulationEnabled` from `true` to `false` (regarding an electrical interface) does not reset the failure simulation settings (e.g., `IsShortToGndAllowed`) of the corresponding signal port to `false` if that property previously was set to `true`.

**ICaDataObject/ICaActiveApplication** If a link is created to a port that is not part of the application, the port will be automatically added to the application.

**ICaRelation/ICaRelations** The following changes affect automation scripts, including the related XPath queries, that automate Bus Manager features and/or access Bus Manager elements.

- The capitalization of the relation name `CommunicationMatricesByECUS` changed to `CommunicationMatricesByEcus`.
- The capitalization of the relation name `CommunicationMatricesByECUSWithProperties` changed to `CommunicationMatricesByEcusWithProperties`.
- The primary role `BusStaticPartISignalIPdu` is divided into two roles, `BusISignalIPdu` (new primary role) and `BusStaticPartIPdu` (secondary role).
- The primary role `BusDynamicPartISignalIPdu` is divided into two roles, `BusISignalIPdu` (new primary role) and `BusDynamicPartIPdu` (secondary role).

- The role names and hierarchy of some ISignal properties changed.
  - Changed role names:
    - `BusISignalToPduMapping` changed to `BusISignalToIPduMapping`
    - `BusCompuMethod` changed to `BusComputationMethod`
    - `BusCompuScale` changed to `BusComputationScale`
  - Changed hierarchy (including changed role names):

    Old Hierarchy

    ```
    BusISignal
    |  BusISignalToPduMapping
    |  |  BusCodedType
    |  |  BusPhysicalType
    |  |  BusCompuMethod
    |  |  |  BusCompuScale
    ```

    New Hierarchy

    ```
    BusISignal
    |  BusISignalToIPduMapping
    |  BusCodedType
    |  BusPhysicalType
    |  BusComputationMethod
    |  |  BusComputationScale
    ```

- The `Sequence number` property of the `BusISignalToIPduMapping` role is obsolete and was removed.
- The role names and hierarchy of some IPDU properties changed.
  - Changed role names:
    - `BusFrameTriggering` changed to `BusFrame`
    - `BusStartingTime` changed to `BusTimeOffset`
    - `BusRepeatingTime` changed to `BusTimePeriod`
    - `BusMinimalDelay` changed to `BusMinimumDelay`
  - Changed hierarchy (including changed role names):

    `BusPduTriggering` was removed from the hierarchy. Additionally, the hierarchy changed as follows:

    Old Hierarchy

    ```
    BusIPdu
    |  BusPduTriggering
    |  |  BusCyclicTiming
    |  |  |  BusStartingTime
    |  |  |  BusRepeatingTime
    |  |  |  BusMinimalDelay
    |  |
    BusEventControlledTiming
    |  |  |  BusRepetitionPeriod
    |  |
    BusCommunicationCluster
    |  |  |  BusPhysicalChannel
    |  |  |  |
    BusFrameTriggering
    ```

    New Hierarchy

    ```
    BusIPdu
    |  BusCyclicTiming
    |  |  BusTimeOffset
    |  |  BusTimePeriod
    |  BusMinimumDelay
    |  BusEventControlledTiming
    |  |  BusRepetitionPeriod
    |  BusCommunicationCluster
    |  |  BusPhysicalChannel
    |  |  |  BusFrame
    ```

- The following property names changed:
  - `Minimal Delay` changed to `Minimum Delay`
  - `Maximal frame length` changed to `Maximum frame length`
  - `Wakeup over bus supported` changed to `Wake up over bus supported`
  - `Pdu router configuration ID` changed to `PDU router configuration ID`
  - `Pdu description` changed to `PDU description`

- `Pdu type` changed to `PDU type`
- `Max delta counter init(ial)` changed to `Max delta counter init`
- `Max no new or repeated data` changed to `Maximum no new or repeated data`
- `Sync counter initial` changed to `Sync counter init`
- `Is start bit Lsb` changed to `Is start bit LSB`
- `Rx accept contained IPDU` changed to `RX accept contained IPDU`
- `Allow Not-A-Number` changed to `Allow not-a-number`
- `Maximal bit length` changed to `Maximum bit length`

**5.5 to 5.6 (dSPACE Release 2016-B)**

The platform management automation API version 1.0 was supported for the last time with ConfigurationDesk 5.5 of dSPACE Release 2016-A.

**5.4 to 5.5 (dSPACE Release 2016-A)**

`ICaApplicationMain: SetCustomInformation` and `ICaComponent: Configure` now return a value. In most cases, this value is `None` (e.g., in Python).

**Note**

Even if the returned value is not used, M-script clients should be aware that a printout follows after calling one of the methods if no semicolon ends the statement. This can cause unexpected output from existing scripts.

**5.3 to 5.4 (dSPACE Release 2015-B)**

Several enumeration values were altered due to changes in the data model.

**Note**

The enumeration changes can cause code from previous releases to malfunction.

**AveragingLevel** The DefinedByModel value was added to the AveragingLevel enumeration. The enumeration now has three values:

- Precise: 1
- Dynamic: 2
- DefinedByModel: 3

**ChannelType**

**Note**

The ChannelType enumeration is obsolete. Do not use it.

**DigitalOutputMode** The values of the DigitalOutputMode enumeration were changed. The values are now:

- Unspecified: -1
- Undefined: 0
- Switch: 1
- LowSideSwitch: 2
- TriState: 3

**Note**

The TriState value is obsolete. Do not use it.

- HighSideSwitch: 4
- PushPull: 8

**HighSideReference** The values of the HighSideReference enumeration were changed. The values are now:

- Individual: 1
- Vbat: 2
- Unused: 3
- Shared: 4
- Shared2: 5

**InitializationMode** The values of the InitializationMode enumeration were changed. They are now:

- First: 1
- Each: 2

**5.2 to 5.3 (dSPACE Release 2015-A)**

The return value of a build process started by the automation now includes a flag DownloadResult to indicate if a download of the real-time application after the build process was successful or not.

**5.1 to 5.2 (dSPACE Release 2014-B)**

**No support of migration from release 2012-B or earlier** Migration of projects and applications which are older than Release 2012-B is no longer supported. HTF, MTF and DTF files are no longer recognized as valid files. Use a ConfigurationDesk version up to 5.1 to migrate these items if necessary.

**TransferType** The enumeration value ExpectedLoadDescription has been changed to LoadRejection. This means only the load rejection settings should be transferred.

**5.0 to 5.1 (dSPACE Release 2014-A)**

Due to the support of multi-processing-unit systems, changes have been made to several areas:

**Platform Management** Since dSPACE Release 2014-A, a platform automation API version supporting multi-processing-unit platforms is available and enabled by default. For backwards compatibility reasons, you can switch back to the old version. Working with the old interfaces requires platform automation API Version 1.0, the new interfaces with extended support of multi-

processing-unit systems require Version 2.0. You can also set the API version via automation interface:

```
Application.Platformmanagement.PlatformAutomationAPIVersion = 1
```

**Scanning registered hardware** Creating a hardware topology by scanning the registered hardware has been changed:

- Use the name of the system to read its inventory and then call the Configure method. For example, if the registered platform has the name SCALEXIO:

```
CreateArguments = [0, "Dummy topology name", "SCALEXIO"]
HardwareTopology.Configure("Create", CreateArguments)
```

- If you want to create a hardware inventory by reading a registered single-processing-unit system, it is possible to use the MAC address of the processing unit, for example:

```
CreateArguments = [0, "Dummy topology name", "11:22:33:44:55:66"]
HardwareTopology.Configure("Create", CreateArguments)
```

**Creating an application process via ICaAlgorithms** When an application process is created via ICaAlgorithms it is now possible to indicate the parent processing unit application. If the parent parameter is null, a new processing unit application will be generated.

```
Algorithms.CreatePreconfiguredApplicationProcessAutomatically([MyModel],
MyProcessingUnitAplication)
```

## 4.4 to 5.0 (dSPACE Release 2013-B)

For dSPACE Release 2013-B, some major changes have been applied to the automation interface. This may lead to scripts from earlier releases to not work correctly with ConfigurationDesk 5.0.

**Switch to Python 2.7** The import of the enumerations module changed:

| Previously | Now |
|---|---|
| `From win32com.client import Enums` | `From dspace.com import Enums` |

**Change of the return type and parameter type of the ICaRelation interface**

- `GetElements` and `GetTopNodes` now return a collection of ICaObjects
- Most of the element methods now have parameters of (C#) type `Object` instead of `ICaDataObject`

**Additional hierarchy level in the application configuration** The processing unit has been added to the application configuration hierarchy in the Executable Application table view. This has the following effects, illustrated by the example of creating an application process:

| | Previously | Now |
|---|---|---|
| `# get the executable application` | `ExeApp = Relation.GetTopNodes().Item(0)` | `ExeApp = Relation.GetTopNodes().Item(0)` |
| `# get processing unit application` | – | `ProcUnitApp = Relation.GetElements(ExeApp).Item(0)` |
| `# create the application process` | `AppProc = ExeApp.CreateChild(ExeApp.DataObjectTypes.Item(0))` | `AppProc = ProcUnitApp.CreateChild(ProcUnitApp.DataObjectTypes.Item(0))` |

**Export of topologies to old XLS file format no longer supported** The export of components like the device topology or the external cable harness to the XLS file format is no longer possible. Use the XLSX file format instead. Even though creating or importing the XLS file format is currently still supported, it is highly recommended to switch to the new format XLSX. The enumeration value for XLS can be used like before.

| Previously | Now |
|---|---|
| `Arguments = []`<br>`Arguments.append(r"c:\MyTopologies\MyExportedDeviceTopology.xls")`<br>`Arguments.append("xls")`<br>`DeviceTopology.Configure("Export", Arguments)` | `Arguments = []`<br>`Arguments.append(r"c:\MyTopologies\MyExportedDeviceTopology.xlsx")`<br>`Arguments.append("xlsx")`<br>`DeviceTopology.Configure("Export", Arguments)` |

# ConfigurationDesk API Reference

**Where to go from here**

Information in this section

# Introduction to ConfigurationDesk's Automation API

## Overview of the API

**Main Parts**

The API of ConfigurationDesk automation is subdivided into three main parts:

- Project and application handling to create, organize, and manipulate projects and applications
- Component handling to create and configure components like the device topology, model topology, or hardware topology
- Data object handling to create or configure data objects or to connect data objects (ports) by creating links between them

**Interface names**

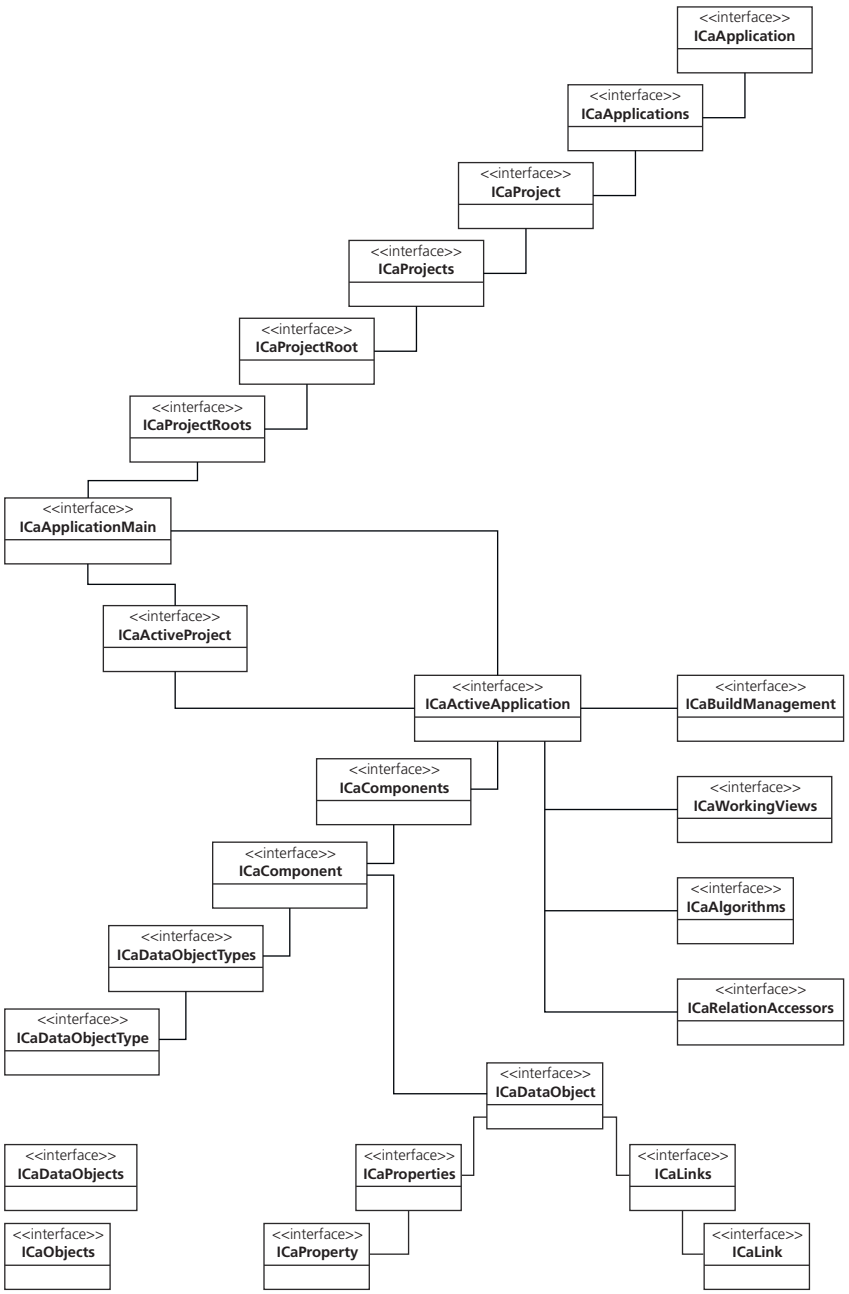
The interface name consists of a prefix, such as ICa, and the object name you know from the Python object in the Python interpreter (for example, ConfigurationDesk's Internal Interpreter).

**Most useful interfaces**

When you start automating ConfigurationDesk, the most useful interfaces to work with are ICaApplicationMain and ICaActiveApplication. ICaApplicationMain is the entry point for all automation work in ConfigurationDesk. It gives you access to project roots, projects and applications. After you create or open a project and a corresponding application, the starting point for actually configuring the signal chain is ICaActiveApplication. From this interface it is possible to get components like the IOFunctionLib, which itself serves as a repository of data objects.

**Overview illustration**

The following chart provides an overview of the automation interface. However, it is not a complete list of all interfaces available in ConfigurationDesk's automation library.



# API Interfaces

**Where to go from here**

Information in this section



# Build Management

**Where to go from here**

Information in this section



## ICaBuildManagement <<Interface>>

**Description**

Provides access to the build management.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| DirectoryName | Gets the folder name relative to the application folder where the build results are located. | Get | *String* |
| Properties | Gets the build properties. | Get | ICaProperties (refer to ICaProperties <<Collection>> on page 197) |

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Build | Builds a real-time application. | None | Result of the build.<br>▪ ICaBuildResult (refer to ICaBuildResult <<Interface>> on page 131) |

1) <Type> **Name**: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaActiveApplication (refer to ICaActiveApplication <<Interface>> on page 203)

## ICaBuildResult <<Interface>>

**Description**

Use this interface to access the results of a build and to download a real-time application.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| Canceled | Gets a value indicating whether the build process was canceled before being started. | Get | *Boolean* |
| Downloaded | Gets a value indicating whether a download of the real time application was successfully performed or not. | Get | *Boolean* |
| ResultFolderFullPath | Gets the full path to the build results folder. | Get | *String* |
| RtaFullPath | Gets the full path to the built real-time application. | Get | *String* |
| Success | Gets a value indicating whether the build process was successful or failed. | Get | *Boolean* |

**Methods**

The element has no methods.

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaBuildManagement (refer to ICaBuildManagement <<Interface>> on page 130)

# Component Handling

**Where to go from here**

Information in this section

## ApplicationProcessToModel <<Enumeration>>

**Description**

This enumeration indicates the relation between models to add and possible new application processes when using the AddModels operation.

**Enumeration values**

The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| OneForEach | For every model added by the call of AddModels create one application process operation. | 0 |
| OneForAll | Create only one application process for all models added by the call of AddModels operation. | 1 |
| No | Create no application process for the models added by the call of AddModels operation. | 2 |

## ConfigurationReplaceMode <<Enumeration>>

**Description**

The mode how to replace a configuration. This enumeration is only for backwards compatibility. A configuration component is no more supported.

**Enumeration values**

The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| DefaultSettings | Replaces a configuration by setting the defaults. | 0 |
| FileImportCds | Replaces a configuration by importing a *.cds file. | 2 |

## DeviceTopologyCreateMode <<Enumeration>>

**Description**

The mode how to create a device topology. The file format changed for ConfigurationDesk 4.3 to *.dtfx.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| EmptyTopology | Creates an empty device topology. | 1 |
| FileImportDtf | Creates a device topology by importing a *.dtfx file. | 2 |
| FileImportXls | Creates a device topology by importing a *.xlsx file. | 3 |

## ExternalWiringCreateMode <<Enumeration>>

**Description** The mode how to create an external wiring. The file format changed for ConfigurationDesk 4.3 to *.echx.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| Calculate | Creates the external wiring by calculating it. | 1 |
| FileImportEch | Creates the external wiring by importing an *.echx file. | 2 |

## HardwareTopologyCreateMode <<Enumeration>>

**Description** The mode how to create a hardware topology. The file format changed for ConfigurationDesk 4.3 to *.htfx.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| ScanRegisteredHardware | Scans the registered hardware to create the topology from. | 0 |
| FileImportHtf | Imports a htfx-file to create the topology from. | 1 |
| EmptyTopology | Creates an empty topology. | 2 |

# ICaAlgorithms <<Interface>>

**Description**

Executes algorithms for a defined set of objects in an active application. Any access through this interface after closing an application can cause unpredictable results.

**Properties**

The element has no properties.

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| AssignChannelSet | Assigns a channel set and a suitable channel to an I/O function block instance or an electrical interface. If parameter AssignChannel is set to False only a channel set will be assigned. If it is not possible to assign the channel set (for example IOFunctionItem is a LambdaDCR) an exception will be thrown. | ▪ <ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)> `IOFunctionItem`: IOFunction instance or electrical interface.<br>▪ <ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)> `ChannelSet`: Channelset to assign.<br>▪ *<Boolean>* `AssignChannels`: Flag to indicate if channel(s) should be assigned. | None |
| AssignDefaultValuesAutomatically | Tries to assign the default values for the given items. Currently supported: FunctionBlocks with assigned hardware. | ▪ *<System.Array>* `Items`: The items to assign the default values for, currently supported are FunctionBlocks | None |
| AssignHardwareAutomatically | Assigns the specified function blocks automatically to hardware resources. | ▪ *<System.Array>* `FunctionBlocks`: The function blocks for which to assign automatically. | None |
| AutoAssignChannels | Assigns suitable channels to an I/O function block instance or an electrical interface. If no channel set was assigned to the given IOFunctionItem before an exception will be thrown. | ▪ <ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)> `IOFunctionItem`: IOFunction instance or electrical interface. | None |
| AutoAssignChannelSet | Assigns a suitable channel set and channels to I/O function block instances or electrical interfaces. | ▪ *<System.Array>* `IOFunctionItems`: Array of IOFunctions or electrical interfaces. | None |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| ConnectIOFunctionBlocksToModelPortBlocks | Lets you automatically map function and model ports of the given function blocks and model port blocks using a name-based algorithm. Ports that cannot be mapped according to mapping rules are ignored. | ▪ <*System.Array*> `Items`: Instances with ports. | Collection containing the new links.<br>▪ ICaLinks (refer to ICaLinks <<Collection>> on page 156) |
| ConnectModelPortBlocksToModelPortBlocks | Lets you automatically map model ports of the given model port blocks for model communication using a namebased algorithm. Ports that cannot be mapped according to mapping rules are ignored. | ▪ <*System.Array*> `Items`: Instances with ports. | Collection containing the new links.<br>▪ ICaLinks (refer to ICaLinks <<Collection>> on page 156) |
| ConnectObjectsAutomatically | Lets you automatically map all ports of a function block and a model port block. A name based algorithm is used to identify matching ports. Ambiguities are automatically resolved if possible. Model port blocks that are not used in the signal chain are automatically added to it. | ▪ <ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)> `First`: ICaDataObject like a function block.<br>▪ <ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)> `Second`: ICaDataObject like a model port block.<br>▪ <*String*> `ConnectionType`: String identifying the algorithm of the connection. Currently only default (empty string) is supported. | Return value of the method.<br>▪ ICaLinks (refer to ICaLinks <<Collection>> on page 156) |
| CreateAssociatedDataObject | Creates an associated ICaDataObject of the specified type for the given ICaDataObject. Currently supported: Create an IOFunctionBlock for the given IOChannel and assign the channel as a hardware resource. | ▪ <ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)> `DataObject`: ICaDataObject to create the new DataObject for (e.g. AnalogOut-IOChannel)<br>▪ <*String*> `AssociatedType`: Name of the new ICaDataObject to create (e.g. 'VoltageOutFunctionBlock')<br>▪ <*String*> `AdditionalInformation`: This parameter is reserved for later use, default is empty string. | The new created ICaDataObject (e.g. a VoltageOutFunctionBlock)<br>▪ ICaDataObject (refer to ICaDataObject <<Interface>> on page 150) |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| CreateInverseBlocks | Lets you create inverse model port blocks for the given model port blocks as counterparts for model communication. Unmapped ports are automatically mapped to the ports of the inverse blocks. | ▪ <*System.Array*> `ModelPortBlocks`: Model port blocks to generate counter parts for. | Collection with created blocks.<br>▪ ICaDataObjects (refer to ICaDataObjects <<Collection>> on page 154) |
| CreatePreConfiguredApplicationProcessAutomatically | Creates an application process for task modeling for every given model implementation automatically. If array is empty an application process will be created for all valid models. If array contains models which are not valid, an error will be generated. | ▪ <*System.Array*> `Models`: The models.<br>▪ <ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)> `Parent`: The processing unit application to create the application process in, default is null for new application process. | None |
| CreateSuitableModelPortBlock | Creates suitable model ports for the specified function blocks. | ▪ <*System.Array*> `FunctionBlocks`: The function blocks to create the model ports for. | None |
| CreateTaskAndRunnableFunctionBlockForIOEvent | Creates for every not used i/o-event of an i/o-function a suitable task and a runnable function block. | ▪ <ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)> `Function`: Function to generate suitable task and runnable function for<br>▪ <ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)> `ApplicationProcess`: Application process, empty for new. | None |
| DeleteCompletely | Deletes ICaDataObject items and usages from the signal chain and from topologies of the active ConfigurationDesk application. For example device: Delete from application and topology. No unresolved elements remain. For example model: Delete model from topology and its assignment to application process and delete assignment of all subelements. If the parameter includes one or more elements which are not deletable an exception will be thrown. | ▪ <*System.Array*> `Entities`: ICaDataObject items to delete. | None |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| ExportConfiguration | Exports the configuration of the active application. Currently supported is an export to a XLSX file. | ▪ <*String*> `FullPath`: The full path to the file. | None |
| ExportConflictsToXML | Exports the conflicts to an XML file. | ▪ <*String*> `FullPath`: The full path to the XML file. Write permission should be granted.<br>▪ <*System.String[]*> `Infos`: Strings to query for special information. Use an empty array for the default information.<br>▪ <*String*> `ConflictGroup`: The conflict group to query for export. Default is String.Empty for all groups. | None |
| ExportPropertyGridToXML | Exports the property grid to XML. | ▪ <*String*> `FullPath`: The full path.<br>▪ <*System.String[]*> `Infos`: Strings to query for special information. Use an empty array for the default information. | None |
| ExportTableViewToXML | Exports a table view to an XML file. | ▪ <*String*> `FullPath`: The full path to the XML file. Write permission should be granted.<br>▪ <*String*> `TableName`: The name of the table to export.<br>▪ <*System.String[]*> `Infos`: Strings to query for special information. Use an empty array for the default information.<br>▪ <*String*> `ColumnsSet`: The set of columns to export. | None |
| GenerateModelInterfaces | Generates model interfaces for the specified model port blocks. If the array of model port blocks is empty, interfaces for all unresolved blocks are generated. | ▪ <*System.Array*> `ModelPortBlocks`: The model port blocks. Empty array for all unresolved blocks. | None |
| GetAllWorkingViewsContainingElements | Calculates all working views in which at least one of the given entities is contained and returns the working view items. If parameter ExcluceGlobal is set to false (default) the global working view is included in the list of | ▪ <*System.Array*> `Entities`: The entities which should be contained into the working views.<br>▪ <*Boolean*> `ExcludeGlobal`: True to exclude it from the returned list, otherwise false. | Array with ICaWorkingView items.<br>▪ *System.Array* |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| | the return working views, otherwise not. | | |
| GetAssignableChannelSets | Returns a collection of assignable channel sets for an I/O function block instance or an electrical interface. If it is not possible to calculate suitable channel sets for the given IOFunctionItem an exception will be thrown. | ▪ <ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)> `IOFunctionItem`: IOFunction instance or electrical interface. | ICaDataObject collection.<br>▪ ICaDataObjects (refer to ICaDataObjects <<Collection>> on page 154) |
| GetConnectedElements | Gets the elements which are connected to the elements in Items. | ▪ <*System.Array*> `Items`: Array of elements to get the connected items for | ICaObjects collection with elements<br>▪ ICaObjects (refer to ICaObjects <<Collection>> on page 158) |
| GetSupportedIOFunctionTypes | Returns the supported and creatable I/O function block types for a channel or channel set If the given item is not a valid channel or channel set an exception will be thrown. | ▪ <ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)> `ChannelItem`: Channel or channel set instance to get IOFunctionTypes for. | Collection of ICaDataObjectTypes<br>▪ ICaDataObjectTypes (refer to ICaDataObjectTypes <<Collection>> on page 155) |
| ImportECUInterfaceContainer | Imports an ECU Interface description container for an ECU Interface Configuration. | ▪ <ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)> `ECUInterfaceConfiguration`: Function instance to import container for.<br>▪ <*String*> `FilePath`: Path to a valid ECU Interface description container. | None |
| OptimizeConfiguration | Sorts the given items automatically. The array contains ICaDataObject items which represent tasks or application processes. For application processes OptimizeConfiguration assigns runnable functions to tasks and determines the execution order of runnable functions within tasks depending on their providersâ€™ model communication connections. If necessary it creates new tasks. Task priorities are set based on their runnable functionsâ€™ restrictions as well as their communication connections. If executed on tasks the operation sorts runnable | ▪ <*System.Array*> `Entities`: ICaDataObject items representing tasks. | None |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| | functions within each task. - If one or more objects are not valid (e.g. null or not of type task or application process) an ArgumentException will be thrown and no sorting will be performed. If the array is empty an ArgumentException will be thrown. | | |
| PropagateToConfigurationDeskModelInterface | Updates the connected model port blocks in ConfigurationDesk for the given function block instances and creates new model port blocks in ConfigurationDesk if the function block has unconnected ports. | ▪ <*System.Array*> `Entities`: Function block instances. | None |
| PropagateToSimulink | Updates the connected model port blocks in ConfigurationDesk for the given function block instances and creates new model port blocks in ConfigurationDesk if the function block has unconnected ports. If ModelName is provided: Updates the Simulink model for the given function block instances and model port blocks.Model port blocks in Simulink are created if necessary.The structure of the model is not modified, e.g., no subsystems are created or removed. No model port blocks are removed from the model. This replaces the former bus operation UpdateConnectedModelPortBlocks. If no ModelName is provided: A new Simulink model containing a structured model interface including model port blocks and subsystems is created. This replaces the former bus operation GenerateStructuredModel. | ▪ <*System.Array*> `Entities`: Function block instances or model port blocks.<br>▪ <*String*> `ModelName`: Name of existing model in ModelTopology, default is null for new model. | None |
| TransferData | Transfers the data specified by theTransferType (refer to TransferType <<Enumeration>> on | ▪ <*System.Array*> `Ports`: The ports.<br>▪ <TransferType (refer to TransferType <<Enumeration>> on | None |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| | page 189)from the ports to the connected ports. | page 189)> `TransferType`: The TransferType enumeration: All (numerical value: 0) AllowedFailureClasses (1) ExpectedLoadDescription (2) | |
| UpdateECUInterfaceContainer | Updates an ECU Interface description container for an ECU Interface Configuration. | ▪ <ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)> `ECUInterfaceConfiguration`: Function instance to update container for.<br>▪ <*String*> `FilePath`: Path to a valid ECU Interface description container. | None |

[1)] <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaActiveApplication (refer to ICaActiveApplication <<Interface>> on page 203)

# ICaComponent <<Collection>>

**Description**

Use this interface to access and configure a specific component. It provides methods to create instances of certainICaDataObject (refer to ICaDataObject <<Interface>> on page 150)s. It is not necessary to create a component explicit. There is always an empty component present after creating an application. With a component and within this interface you can configure (including creating) the component and create one or more root objects for it. The supported types of root objects can be examined by the*DataObjectTypes(ICaComponent)*property. To check whether an operation is supported by a component, you can call the *GetSupportedOperations(ICaComponent)*method. For a brief description of the components and their supported operations see below. - The implementing automation object for this interface is only valid in its active application. Any access after closing the application is undefined and can cause unpredictable results.

| ModelTopology | Description |
|---|---|
| Creating a ModelTopology | Creating a model topology is done by calling the*Configure(ICaComponent)*method with the appropriate parameters. A model topology can be created by importing either a Simulink (*.mdl, *.slx) file, a model topology (*.mtfx) file or a multi model configuration (*.mcd) file. In all cases the call of the configure method requires first the name of the operation ("Create") and second an array with the specific information. Additionally it is possible to create an empty model topology. In some cases it may be useful to create a |

| ModelTopology | Description |
|---|---|
| | new model topology with models which should be assigned automatically to a new application process. Therefore it is possible to indicate that models imported by creating an new model topology should be assigned to new application processes. Because an application process is associated to a processing unit application the possibility to create a new application process depends on how many processing unit applications are present in the active application. Is no processing unit application present, a new one will be created to which the new application processes will be assigned. If only one processing unit application is present the new application processes are assigned to it. If more than one processing unit application is present the create operation will fail.<br><br>To create a model topology by importing a Simulink (*.mdl) file, you need to specify a flag for the import (use "1" or use theFileImportMdl(ModelTopologyCreateMode) (refer to ModelTopologyCreateMode <<Enumeration>> on page 175)type), the name of the model topology, the full path to the file to import, a Boolean flag specifying if model should be analyzed immediately and the model analyze command (empty string if not necessary or not wanted). It is possible to indicate that a new application process should be created for the imported model if no or only one processing unit application is present in the active application. Use an optionally Boolean flag "true". Analyze immediately must be set to true to support this.<br><br>To create a model topology by importing a model topology file (*.mtfx), you need to specify a flag for the import (use "2" or use theFileImportMtf(ModelTopologyCreateMode) (refer to ModelTopologyCreateMode <<Enumeration>> on page 175)type), the name of the model topology, the full path to the file to import and a Boolean flag specifying if model should be analyzed immediately. It is possible to indicate that new application processes should be created for the imported models if no or only one processing unit application is present in the active application. Use an optionally Boolean flag "true". Analyze immediately must be set to true to support this.<br><br>To create a model topology by importing a multi model configuration file (*.mcd), you need to specify a flag for the import (use "3" or use theFileImportMcd(ModelTopologyCreateMode) (refer to ModelTopologyCreateMode <<Enumeration>> on page 175)type), the name of the model topology, the full path to the file to import and a Boolean flag specifying if all models contained in the topology should be analyzed immediately. It is possible to indicate that new application processes should be created for the imported models if no or only one processing unit application is present in the active application. Use an optionally Boolean flag "true". Analyze immediately must be set to true to support this. If the user wants to create an application process for every model the optional parameter CreateMultipleApplicationProcesses can be used (default is True). This parameter will only be evaluated if the preceding parameter is set to True.<br><br>To create an empty model topology, you need to specify a flag (use "4" or use theEmptyTopology(ModelTopologyCreateMode) (refer to ModelTopologyCreateMode <<Enumeration>> on page 175)type) and the name of the model topology.<br><br>To create a model topology by importing a Simulink (*.Slx) file, you need to specify a flag for the import (use "5" or use theFileImportSlx(ModelTopologyCreateMode) (refer to ModelTopologyCreateMode <<Enumeration>> on page 175)type), the name of the model topology, the full path to the file to import, a Boolean flag specifying if model should be analyzed immediately and the model analyze command (empty string if not necessary or not wanted). It is possible to indicate that a new application process should be created for the imported model if no or only one processing unit application is present in the active application. Use an optionally Boolean flag "true". Analyze immediately must be set to true to support this. |

| ModelTopology | Description |
|---|---|
| | To create a model topology by importing an other file type (possible V-ECU *.ctlgz) file, you need to specify a flag for the import (use "6" or use theFileImportOther(ModelTopologyCreateMode) (refer to ModelTopologyCreateMode <<Enumeration>> on page 175)type), the name of the model topology and the full path to the file to import. It is possible to indicate that new application processes should be created for the imported models if no or only one processing unit application is present in the active application. Use an optionally Boolean flag "true". |
| Replacing a Model Topology | Replacing a model topology is done in the same way as creating one, other than you have to change the first parameter of the*Configure(ICaComponent)*method to "Replace". |
| Removing a Model Topology | Removing a model topology is done by calling the*Configure(ICaComponent)*method with the "Remove" operation and an empty array. The component then will be cleared but will be still valid and empty. |
| Renaming a Model Topology | Renaming a model topology is no more supported. |
| Save as for a Model Topology | You can save a model topology as an *.mtfx file by calling the*Configure(ICaComponent)*method with the "SaveAs" operation. The array should contain only the full path to the *.mtfx file to save. |
| Analyzing a Model | Call the "Analyze" operation with the*Configure(ICaComponent)*method and an empty array or call the "AnalyzeComplete" operation to analyze model interfaces with task configuration data (and initialization). |
| Set MATLAB initialize Command | Set the initialize command by calling the "SetCommand" operation with the*Configure(ICaComponent)*method and an array that contains the initialization command as a string. |
| Add a model | Add a model to the model topology by calling the "AddModel" operation with the*Configure(ICaComponent)*method. The parameter array should contain 1. full path to the model file 2. true or false for the option to analyze immediately 3. matlab initialization command or empty string 4. true or false for the option to create a preconfigured application process. An optional parameter (5.) contains the name or the automation ICaDataObject representing the desired application process to assign the model to. If parameter 5 is given the parameter 4 must be set to false. If parameter 4 is set to true or an application process with the given name cannot be determined or the given ICaDataObject does not represent an valid application process an exception will be thrown. |
| Remove a model | Remove a model from the model topology by calling the "RemoveModel" operation with the*Configure(ICaComponent)*method. The parameter array should contain only the name of the model to remove. |
| Replace a model | Replace a model from the model topology by calling the "ReplaceModel" operation with the*Configure(ICaComponent)*method. The parameter array should contain the path to the new model, the flag if model should be analyzed completely, a string for the model initialization command (possible empty) and the name of the model to replace. If the operation succeeds the new model will be returned as content in the result array. |
| Add multiple models | To add more than one model at once the "AddModels" operation can be used with the*Configure(ICaComponent)*method. To indicate if and how to create application processes for the models you have to use theApplicationProcessToModel (refer to ApplicationProcessToModel <<Enumeration>> on page 133)enumeration. To provide different information for the different models you have to create anICaModelDescription (refer to ICaModelDescription <<Interface>> on page 157)object with the "CreateModelDescription" operation for every model to add. Call the operation with: Configure("AddModels", [ApplicationProcessToModel-EnumerationValue, ModelDescription1, ModelDescription2, ...]) |

| ModelTopology | Description |
|---|---|
| CreateModelDescription | The CreateModelDescription operation must be used to create an object which allows to provide information about a model to add to the application with the "AddModels operation. Call the operation with Configure("CreateModelDescription", []) |
| Reload an mcd file | Reload an mcd file by calling the "ReloadMCD" operation with the*Configure(ICaComponent)*method. The parameter array should be empty. |
| Clear an mcd file | Clear an mcd file by calling the "ClearMCD" operation with the*Configure(ICaComponent)*method. The parameter array should be empty. |
| Update the model implementation | Update the model implementation by calling the "UpdateModelImplementation" operation with the*Configure(ICaComponent)*method. The "UpdateModelImplementation" method replaces the "UpdateVEcuImplementation" method which only updates V-ECUs. Use an empty parameter array for all model implementations which can be udpated or use the parameter array to specify the model implementation by their names or by their automation object. |
| Skip model code generation | Skip the generation of model code by calling the "SkipModelBuild" operation with the*Configure(ICaComponent)*method. The parameter array should contain true or false to skip the generation or not. In no model name is given in the parameter array the model code generation will be skipped for all models. |
| Generate model code | Generate model code by calling the "GenerateModelCode" operation with the*Configure(ICaComponent)*method. The parameter array should contain the name of the models to generate model code for. If the parameter is empty model code will be generated for all possible models. |

| HardwareTopology | Description |
|---|---|
| Creating a Hardware Topology | Creating a hardware topology is done by calling the*Configure(ICaComponent)*method with the appropriate parameters. There are three ways to create a hardware topology: by scanning a registered hardware system, by importing an *.htfx file or by creating an empty topology. In all three cases, the call of the configure method requires the name of the operation ("Create") and an array with the specific information.<br><br>To create a hardware topology by scanning a registered hardware system for the arguments array, you have to specify the flag for the creation mode (use "0" or use theScanRegisteredHardware(HardwareTopologyCreateMode) (refer to HardwareTopologyCreateMode <<Enumeration>> on page 134)type) first. Add the display name for the topology (for backwards compatibility only) and the name of the system. If the system is not a multi Processing Unit system it is possible to search by an IPAddress number (string value) or a MAC address (colon separated string) instead by the name.<br><br>To create a hardware topology by importing a hardware topology file (*.htfx), you need to specify a flag for the import (use "1" or use theFileImportHtf(HardwareTopologyCreateMode) (refer to HardwareTopologyCreateMode <<Enumeration>> on page 134)type), Add the display name of the topology and the full path to the file to import.<br><br>To create an empty hardware topology, specify the flag for empty (use "2" or use theFileImportHtf(HardwareTopologyCreateMode) (refer to HardwareTopologyCreateMode <<Enumeration>> on page 134)type), and the display name which is needed for the arguments array. |
| Replacing a Hardware Topology | Replacing a hardware topology is done in the same way as creating one, other than you have to change the first parameter of the*Configure(ICaComponent)*method to "Replace". |

| HardwareTopology | Description |
|---|---|
| Removing a HardwareTopology | Removing a hardware topology is done by calling the*Configure(ICaComponent)*method with the "Remove" operation and an empty array. The component then will be cleared but will be still valid and empty. |
| Renaming a HardwareTopology | Renaming a hardware topology is no more supported. |
| Save as for a HardwareTopology | You can save a hardware topology as an *.htfx file by calling the*Configure(ICaComponent)*method with the "SaveAs" operation. The array should contain only the full path to the *.htfx file to save. |

| DeviceTopology | Description |
|---|---|
| Creating a Device Topology | Creating a device topology is done by calling the*Configure(ICaComponent)*method with the appropriate parameters. There are three ways to create a DeviceTopology: by importing a *.dtfx file, by importing an *.xlsx file or by creating an empty topology. In all three cases, the call of the configure method requires the name of the operation ("Create") and an array with the specific information.<br>To create a device topology by importing a *.dtfx file, you have to specify the flag for the creation mode (use "2" or use theFileImportDtf(DeviceTopologyCreateMode) (refer to DeviceTopologyCreateMode <<Enumeration>> on page 133)type) first. Add the display name for the topology and the full path to file to import.<br>To create a device topology by importing an Excel (*.xlsx) file, you need to specify a flag for this import (use "3" or use theFileImportXls(DeviceTopologyCreateMode) (refer to DeviceTopologyCreateMode <<Enumeration>> on page 133)type), Add the display name of the topology and the full path to the file to import. Note: it is only possible to create a device topology by importing an Excel file if Excel is installed on the system.<br>To create an empty device topology, only the flag for empty (use "1" or use theEmptyTopology(DeviceTopologyCreateMode) (refer to DeviceTopologyCreateMode <<Enumeration>> on page 133)type) and the display name is needed for the arguments array. |
| Replacing a Device Topology | Replacing a device topology is done in the same way as creating one, other than you have to change the first parameter of the*Configure(ICaComponent)*method to "Replace". It is not possible to replace a device topology by importing an Excel file. |
| Removing a Device Topology | Removing a device topology is done by calling the*Configure(ICaComponent)*method with the "Remove" operation and an empty array. The component then will be cleared but will be still valid and empty. |
| Renaming a Device Topology | Renaming a device topology is no more supported. |
| Save as for a Device Topology | You can save a device topology as a *.dtfx file by calling the*Configure(ICaComponent)*method with the "SaveAs" operation. The array should contain only the full path to the *.dtfx file to save. |
| Adding a Device Topology | You can add a device topology in the same way as creating a new one by calling the*Configure(ICaComponent)*method with exactly the same arguments array and the "Add" operation. |
| Exporting a Device Topology | You can export a device topology as an *.xlsx file by calling the*Configure(ICaComponent)*method with the "Export" operation. The parameter array must contain the full path to the *.xlsx file to save. Note: For this operation Excel must be installed on the system. |

| ExternalWiring | Description |
|---|---|
| Creating an External Wiring | Creating an external wiring is done by calling the*Configure(ICaComponent)*method with the appropriate parameters. There are two ways to create an ExternalWiring: by importing an *.echx file or by calculating it.<br>To create an external wiring by importing an *.echx file, you have to specify the flag for the creation mode (use "2" or use theFileImportEch(ExternalWiringCreateMode) (refer to ExternalWiringCreateMode <<Enumeration>> on page 134)type) first. Add the display name for the topology and the full path to the file to import.<br>To create an external wiring by calculating it, you have to to specify a flag for the calculation (use "1" or use theCalculate(ExternalWiringCreateMode) (refer to ExternalWiringCreateMode <<Enumeration>> on page 134)type), Then add the display name of the topology. |
| Replacing an External Wiring | Replacing an external wiring is done in the same way as creating one, other than you have to change the first parameter (the operation string) of the*Configure(ICaComponent)*method to "Replace". |
| Removing an External Wiring | Removing an external wiring is done by calling the*Configure(ICaComponent)*method with the "Remove" operation and an empty array. The component then will be cleared but will be still valid and empty. |
| Renaming an ExternalWiring | Renaming an external wiring is no more supported. |
| Save as for an External Wiring | You can save an external wiring as an *.echx file by calling the*Configure(ICaComponent)*method with the "SaveAs" operation. The array should contain only the full path to the *.echx file to save. |
| Calculating an External Wiring | You can calculate an external wiring by calling the*Configure(ICaComponent)*method with "Fill" as the operation string and an empty array. The method does not check if an external wiring is present in the application. |
| Exporting an External Wiring | You can save an external wiring as an *.xlsx file by calling the*Configure(ICaComponent)*method with the "Export" operation. The parameter array must contain the full path to the *.xlsx file to save Note: For this operation Excel must be installed on the system. |

| IOFunctionLib | Description |
|---|---|
| Configuring the IOFunctionLib | It is not possible to configure the IOFunctionLib component even it is not necessary to create or remove it. Therefore the call of the*GetSupportedOperations(ICaComponent)*method always returns an empty collection of strings. |

| BusManager | Description |
|---|---|
| Configuring the Bus Manager | It is possible to configure the Bus Manager component but it is not necessary to create or remove it. |
| Importing a communication matrix | Import a communication matrix by calling the*Configure(ICaComponent)*method with the "AddCommunicationMatrix" operation. With this operation, you can import a communication matrix from a file (*.xml, *.arxml, *.dbc,*.ldf) into the current application. It is only possible to load a specific communication matrix once into an application. |
| Removing a communication matrix | Remove a communication matrix by calling the*Configure(ICaComponent)*method with the "RemoveCommunicationMatrix" operation and one of the following string values: Use "True" to remove the selected communication matrix and all its elements that are assigned to bus configurations. Use "False" to remove only the communication matrix. If elements of the communication matrix are assigned to bus configurations, these elements are marked as unresolved. |

| BusManager | Description |
|---|---|
| Assigning an element to a bus configuration | Assign elements from a communication matrix by calling the*Configure(ICaComponent)*method with the "AssignElements" operation. With this operation, you can add elements from a communication matrix to a specific bus configuration. Bus configurations can contain elements from different communication matrices. |
| Removing an element from a bus configuration | Remove elements by calling the*Configure(ICaComponent)*method with the "RemoveElements" operation. With this operation, you can remove elements from bus configurations. Those elements can be of different bus configurations. |
| Replacing assigned communication matrices | Replace an assigned communication matrix by calling the*Configure(ICaComponent)*method with the "ReplaceCommunicationMatrixAssignment" operation. With this operation, you can replace a communication matrix in a specific bus configuration (first argument) with a modified version of the matrix (second argument). |
| Get available features | Get a list of all the features (used and unused) that are available for the selected bus configuration element by calling the Configure(ICaComponent) method with the "GetAvailableFeatures" operation. The list provides the role names that are required to add an available feature via the "AddFeature" operation. |
| Add feature | Add an available feature to the selected bus configuration element by calling the Configure(ICaComponent) method with the "AddFeature" operation. To add an available feature, you must use its role name as a string value. To get an overview of the role names of the available features, you can use the "GetAvailableFeatures" operation. |
| Adding elements to communication matrices | Add a new communication matrix element (first argument) to an existing element (second argument) by calling the*Configure(ICaComponent)*method with the "AddElementToCommunicationMatrix" operation. To add a new element, you must use its role name as a string value. Depending on the existing element, it is possible to call this configure operation several times to add more than one new element. |
| Undo modifications of communication matrices | Undo the modifications of a communication matrix by calling the*Configure(ICaComponent)*method with the "UndoChangesToCommunicationMatrix" operation in one of the following ways: For a specific imported communication matrix element (first argument), undo the changes to some of its properties (second argument, list of properties) or to all of its properties (omit second argument). In case of an added communication matrix element (first argument), you can only remove the element and all its dependent elements (omit second argument). For the entire communication matrix, you can undo all of the modifications (empty first argument, omit second argument). |
| Generating bus simulation | Generate bus simulation containers by calling the*Configure(ICaComponent)*method with the "GenerateContainers" operation. |
| Adding frame captures | Add a frame capture to the Inspection part of a specific bus configuration by calling the*Configure(ICaComponent)*method with the "AddFrameCapture" operation. It is possible to call this configure operation several times to add more than one frame capture to the Inspection part. |
| Adding frame gateways | Add a frame gateway of the given type (first argument) to the Gateways part of a specific bus configuration (second argument) by calling the*Configure(ICaComponent)*method with the "AddGateway" operation. It is possible to call this configure operation several times to add more than one frame gateway to the Gateways part. Currently supported gateway type is "BusFrameGateway". |
| Adding filter rules | Add a filter rule to a specific frame capture filter or frame gateway filter by calling the*Configure(ICaComponent)*method with the "AddFilterRule" operation. It is possible to call this configure operation several times to add more than one filter rule to the specific filter. |

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| ComponentType | Gets the type of the component. | Get | *String* |
| Count | Returns the number of data objects of the components data repository. | Get | *Signed 32 Bit Integer* |
| DataObjectTypes | Gets the collection of creatable data object types of this component. The items can be used as type parameter when creatingICaDataObject (refer to ICaDataObject <<Interface>> on page 150)s. | Get | ICaDataObjectTypes (refer to ICaDataObjectTypes <<Collection>> on page 155) |
| IsConfigured | Returns always true. | Get | *Boolean* |

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Configure | Provides access to the basic creation of a component and to configure various aspects of it. The supported configuration possibilities can be retrieved by the GetSupportedOperations method. | ▪ <*String*> `operation`: The operation.<br>▪ <*System.Array*> `Params`: The params. | None |
| Contains | Tests if a data object is contained in the component data repository. | ▪ <ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)> `DataObject`: Data object instance searched for. | True if the repository contains the data object.<br>▪ *Boolean* |
| CreateRootObject | Creates a top levelICaDataObject (refer to ICaDataObject <<Interface>> on page 150)instance of the type given by the parameter "ICaDataObjectType". A newly created object will become a member of the component's data. repository. | ▪ <ICaDataObjectType (refer to ICaDataObjectType <<Interface>> on page 154)> `Type`: Type of the data object.<br>▪ <*String*> `Name`: Name of the data object or auto-generated name if not specified or is empty string. | New data object.<br>▪ ICaDataObject (refer to ICaDataObject <<Interface>> on page 150) |
| GenerateWorkingViews | Generates WorkingViews according to the content of the topology. If a component does not support generating WorkingViews a not supported exception will be thrown. | ▪ <*Signed 32 Bit Integer*> `Mode`: Parameter is for further use, default is 0 | None |
| GetSupportedOperations | Gets a collection of strings that identifies the supported operation by this component. Collection can be empty. | None | String collection with supported operations.<br>▪ ICaStrings (refer to ICaStrings <<Collection>> on page 165) |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Item | Returns a data object from the components data repository according to the specified index or full path (full path currently not supported). | ▪ *<String>* `Index`: Full path of the data object or a numeric index value. | The found data object.<br>▪ ICaDataObject (refer to ICaDataObject <<Interface>> on page 150) |

1) <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaComponents (refer to ICaComponents <<Interface>> on page 149)

## ICaComponents <<Interface>>

**Description**

Provides access to the components of an application. The collection contains predefinedICaComponent (refer to ICaComponent <<Collection>> on page 141)s and is read-only. The implementing object is only valid in its active application. Any access after closing the application is undefined and can cause unpredictable results.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| BusManager | Gets the bus manager topology. | Get | ICaComponent (refer to ICaComponent <<Collection>> on page 141) |
| DeviceTopology | Gets the device topology. | Get | ICaComponent (refer to ICaComponent <<Collection>> on page 141) |
| ExternalWiring | Gets the external wiring topology. | Get | ICaComponent (refer to ICaComponent <<Collection>> on page 141) |
| HardwareTopology | Gets the hardware topology. | Get | ICaComponent (refer to ICaComponent <<Collection>> on page 141) |
| IOFunctionLib | Gets the function library topology. | Get | ICaComponent (refer to ICaComponent <<Collection>> on page 141) |

| Name | Description | Get/Set | Type |
|---|---|---|---|
| ModelTopology | Gets the model topology. | Get | ICaComponent (refer to ICaComponent <<Collection>> on page 141) |

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Contains | Tests if a component with the specified type name is contained in the collection. | ▪ *<String>* `Type`: Configuration type | True if the collection contains a component with given type.<br>▪ *Boolean* |
| Item | Returns a component according to the specified index or type. | ▪ *<String>* `Index`: Type of the component or a numeric index value. | The found component object.<br>▪ ICaComponent (refer to ICaComponent <<Collection>> on page 141) |

1) <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaActiveApplication (refer to ICaActiveApplication <<Interface>> on page 203)

# ICaDataObject <<Interface>>

**Description**

Provides access to a DataObject that specifies a logical element within anICaComponent (refer to ICaComponent <<Collection>> on page 141)application. The implementing object is only valid in the current application. Any access after closing the application or after deleting the object is undefined and can cause unpredictable results. A data object can have the following membership states:

- Application AND Component Data Repository: Object is both in application and repository.
- Application ONLY: Object is only in the application and not in the repository. Usually this means the object is unresolved.
- Component Data Repository ONLY: Object is only in the repository and not in the application.
- None: Object is neither in the repository nor in the application. This usually means the object is no longer valid.

The membership states are changed by modifying the IsInApplication and IsInRepository properties if this is allowed by the real object. Therefore, the

behavior by setting these properties depends on the component type that the object belongs to and on the type of the ICaDataObject (for example, port, function). The IsInApplication property has the following constraints regarding the different topologies.

| IsInApplication | Description |
|---|---|
| Device Topology | All elements of the device topology support this property. it is possible to set the property to True or False. It is also possible to get the value. By setting the IsInApplication property to True, a whole device on its own can be added to or removed from the application. If a subelement is contained in the application, the corresponding parent objects return True for this property too. |
| Model Topology | All elements of the model topology support this property. It is possible to set the property to True or False. It is also possible to get the value. By setting the IsInApplication property to True, a whole model on its own can be added to or removed from the application. If a subelement is contained in the application, the corresponding parent objects return True for this property too. |
| IOFunctionLib | This property is not supported by all elements of the IOFunctionLib and not in the same way. Only function types can be added (instantiated) to the application by setting this property to True. This can be done in a loop to instantiate a function type more than once. Elements of a higher level always return False for this property. Setting their property to True results in an exception. Setting this property to False on instantiated functions deletes the instance. Setting this property on function ports has no effect. Access to the automation object after deleting the corresponding element in the ConfigurationDesk application is undefined and can result in an exception. |
| Hardware Topology | On elements of the hardware topology, the property IsInApplication is the same as the assigned or unassigned state. Because it is not possible to assign one specific element with an automation task, an assignment is forbidden. (For assignment use theAssignHardwareAutomatically(ICaAlgorithms) (refer to ICaAlgorithms <<Interface>> on page 135)method.) Setting this property to True therefore results in an exception. Only the removal of an assignment (setting the property to False) and the getting of the property are allowed. If a subelement is assigned, the corresponding parent objects return True for this property too. |
| External Wiring | There are currently no elements that support this property. |

The IsInRepository property corresponds to the element state "unresolved". Setting this property is only supported by elements of the device topology and the hardware topology and only for the value False. Setting the value to True or setting the value of elements from other topologies results in an exception. The IsInRepository property has the following constraints regarding the different topologies.

| IsInRepository | Description |
|---|---|
| Device Topology | Getting the value of the property and setting it to False is supported. If a subelement is unresolved, the corresponding parent objects return False for this property too. |
| Model Topology | Only getting the value of the property is supported. Setting the IsInApplication property to False for an unresolved (IsInRepository=false) element invalidates this element and all its subelements. |
| IOFunctionLib | Only getting the value of the property is supported. For non-instances (function types) the return value is always True. Instances from plug-ins that were not found during startup of ConfigurationDesk have the state unresolved. They return True for IsInApplication and False for IsInRepository. Subblocks and ports from these unresolved functions are not always unresolved. The behavior of such objects is undefined. |

| IsInRepository | Description |
|---|---|
| Hardware Topology | Getting the value from this property is supported. Data objects like System, Rack or Board can be removed from the repository by setting the property to False. It is not possible to remove single channels from the repository. If one channel is unresolved, the whole corresponding board is unresolved too. |
| External Wiring | There are currently no elements that support this property. |

**Properties** The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| ConnectedObjects | Gets the collection of directly connected ICaDataObject's. | Get | ICaDataObjects (refer to ICaDataObjects <<Collection>> on page 154) |
| DataObjectTypes | Gets the collection of creatable data object types that are supported by this data object. Empty collection if no type is supported. The items can be used as type parameters when creating elements. | Get | ICaDataObjectTypes (refer to ICaDataObjectTypes <<Collection>> on page 155) |
| ImplementingType | Returns the name of the interface that is implemented by this object. | Get | *String* |
| IsInApplication | Gets or sets a value indicating whether this instance is a member of the current application. Changing this value adds/removes the data object to/from the current application if the type of the data object and the topology supports this. | Get/Set | *Boolean* |
| IsInRepository | Gets or sets a value indicating whether this instance is a member of a certain component's data repository. Changing this value adds/removes the data object to/from the component's data repository if the type of the data object and the topology supports this. True for a newly created data object. | Get/Set | *Boolean* |
| Links | Gets the collection ofICaLink (refer to ICaLink <<Interface>> on page 156)s. | Get | ICaLinks (refer to ICaLinks <<Collection>> on page 156) |
| Name | Gets or sets the name of the object. | Get/Set | *String* |
| Parent | Gets the parent data object. Null if root. | Get | *ICaDataObject* |
| Properties | Gets the properties of the object. | Get | ICaProperties (refer to ICaProperties <<Collection>> on page 197) |
| Roles | Gets the roles which are currently associated with the ICaDataObject. This preliminary and experimental feature is not fully documented. | Get | ICaStrings (refer to ICaStrings <<Collection>> on page 165) |
| Type | Gets the data object type. | Get | ICaDataObjectType (refer to ICaDataObjectType |

| Name | Description | Get/Set | Type |
|---|---|---|---|
| | | | <<Interface>> on page 154) |

## Methods

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Contains | Tests if Object is contained in the children's collection. | ▪ *<ICaDataObject>* `DataObject`: The data object. | True if contained in the children's collection, otherwise false.<br>▪ *Boolean* |
| CreateChild | Creates an*ICaDataObject*instance of a certain type as a child of the current data object. The*IsInRepository(ICaDataObject)*value of the created child data object value is inherited from the parent value. | ▪ <ICaDataObjectType (refer to ICaDataObjectType <<Interface>> on page 154)> `Type`: Type of the data object.<br>▪ *<String>* `Name`: Name of the data object or auto-generated name if not specified or is empty string. | New data object.<br>▪ *ICaDataObject* |
| Equals | Checks whether or not the internal referenced object of the given ICaDataObject refers to the same internal referenced object of this instance. Different to the call of Equals the '==' operator checks equality of the wrapping automation objects. | ▪ *<ICaDataObject>* `DataObject`: ICaDataObject to compare with. | True if equal, otherwise false.<br>▪ *Boolean* |
| GetCount | Gets the number of ICaDataObjects in the children's collection. | None | Return value of the method.<br>▪ *Signed 32 Bit Integer* |
| IsOfRole | Returns true if the given role is currently associated with the ICaDataObject, otherwise false. This preliminary and experimental feature is not fully documented. | ▪ *<String>* `Role`: String which identifies role to query for. | True if role is currently associated with the object, otherwise false.<br>▪ *Boolean* |
| Item | Returns a child data object according to the specified index or name. | ▪ *<String>* `Index`: Name of the data object or a numeric index value. | The found data object.<br>▪ *ICaDataObject* |

1) <Type> `Name`: Description

## Returned by

The element is returned by properties or methods of the following elements:

- ICaAlgorithms (refer to ICaAlgorithms <<Interface>> on page 135)
- ICaComponent (refer to ICaComponent <<Collection>> on page 141)
- ICaDataObjects (refer to ICaDataObjects <<Collection>> on page 154)

- ICaLink (refer to ICaLink <<Interface>> on page 156)
- ICaRelation (refer to ICaRelation <<Interface>> on page 158)
- ICaWorkingView (refer to ICaWorkingView <<Collection>> on page 168)

## ICaDataObjects <<Collection>>

**Description**

Provides access to ICaDataObjects. The collection of the implementing object is not updated if changes like removing or adding objects occurs. Therefore any access after changes can cause unpredictable results.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| Count | Returns the number of data objects in the collection. | Get | *Signed 32 Bit Integer* |

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Contains | Tests if a data object with the specified name is contained in the collection. | ▪ <ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)> `DataObject`: The data object to check for. | True if the collection contains the given data object.<br>▪ *Boolean* |
| Item | Returns a data object according to the specified index or type. | ▪ *<String>* `Index`: Name of the data object or a numeric index value. | The found data object.<br>▪ ICaDataObject (refer to ICaDataObject <<Interface>> on page 150) |

1) <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaAlgorithms (refer to ICaAlgorithms <<Interface>> on page 135)
- ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)

## ICaDataObjectType <<Interface>>

**Description**

Provides information about the type of a creatable data object.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| Name | Returns the name of the data object type. | Get | *String* |

**Methods**

The element has no methods.

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)
- ICaDataObjectTypes (refer to ICaDataObjectTypes <<Collection>> on page 155)

## ICaDataObjectTypes <<Collection>>

**Description**

Provides access to creatable data object types that can be used for signal chain or topology configuration. Access to the implementing object after closing the application or removing the corresponding data object can cause unpredictable results.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| Count | Returns the number of data object types. | Get | *Signed 32 Bit Integer* |

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Item | Returns a data object type according to the specified index or name. | ▪ *<String>* `Index`: Name of the data object type or a numeric index value. | The found data object type.<br>▪ ICaDataObjectType (refer to ICaDataObjectType <<Interface>> on page 154) |

1) <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaAlgorithms (refer to ICaAlgorithms <<Interface>> on page 135)
- ICaComponent (refer to ICaComponent <<Collection>> on page 141)

- ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)
- ICaRelation (refer to ICaRelation <<Interface>> on page 158)

## ICaLink <<Interface>>

**Description**

Provides access to the application global link data of a connection. Access to the implementing object after changing or removing the link or the corresponding data objects is undefined and can cause unpredictable results.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| First | Gets the first data object of the link. Access to it after deleting a link or port is undefined. | Get | ICaDataObject (refer to ICaDataObject <<Interface>> on page 150) |
| ImplementingType | Returns the name of the automation interface that is implemented by this object. | Get | *String* |
| Second | Gets the second data object of the link. Access to it after deleting a link or port is undefined. | Get | ICaDataObject (refer to ICaDataObject <<Interface>> on page 150) |

**Methods**

The element has no methods.

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaActiveApplication (refer to ICaActiveApplication <<Interface>> on page 203)
- ICaLinks (refer to ICaLinks <<Collection>> on page 156)

## ICaLinks <<Collection>>

**Description**

Provides access to links. The implemented collection of links is not updated after changes. Access to links in the collection after changing the links is undefined.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| Count | Returns the number of links. The collection can be empty. | Get | *Signed 32 Bit Integer* |

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Item | Returns a link according to the specified index. | ▪ *<Signed 32 Bit Integer>* `Index`: Numeric index value. | The found link.<br>▪ ICaLink (refer to ICaLink <<Interface>> on page 156) |

1) <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaAlgorithms (refer to ICaAlgorithms <<Interface>> on page 135)
- ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)

# ICaModelDescription <<Interface>>

**Description**

An implementation of this interface can be used to provide information about a model to add to the application using the AddModels operation. To get an implementation of this interface use the "CreateModelDescription" operation of the model topology.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| AnalyzeComplete | Gets or sets a flag to indiciate if the model should be analyzed completely. Default is true. Flag is ignored for not supported types. | Get/Set | *Boolean* |
| FilePath | Gets or sets the file path to the model. | Get/Set | *String* |
| InitializeCommand | Gets or sets the model initialization command for Matlab. Default is empty. Command is ignored for not supported types. | Get/Set | *String* |

**Methods**

The element has no methods.

# ICaObjects <<Collection>>

**Description**

Provides access to a collection of arbitrary objects (for example, currently selected objects like ICaDataObject, ICaWorkingView and ICaLink). The collection does not reflect changes made by adding or deleting objects after getting this automation object. Therefore any access after such changes is undefined and can cause unpredictable results.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| Count | Returns the number of objects in the collection. | Get | *Signed 32 Bit Integer* |

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Item | Returns an object according to the specified index in the collection. | ▪ <*Object*> `Index`: Index of object in the collection. Attention: if given index is a string ensure that the objects in the collection supports name property. | None |

1) <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaActiveApplication (refer to ICaActiveApplication <<Interface>> on page 203)
- ICaAlgorithms (refer to ICaAlgorithms <<Interface>> on page 135)
- ICaRelation (refer to ICaRelation <<Interface>> on page 158)

# ICaRelation <<Interface>>

**Description**

This interface provides access to elements with a specific relation by using a RelationAccessor. A RelationAccessor represents a specific relation between elements of the application. Given an element `"a"` you can access all the elements with this specific relation related to `"a"`. Not all methods described in this interface are supported by all RelationAccessors.The RelationAccessor itself can be queried from the ICaRelations (refer to ICaRelations <<Collection>> on page 160).

**Properties** The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| Name | Gets the name of the Relation. | Get | *String* |

**Methods** The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| AddElements | Adds the given elements to the given Object. | ▪ <*Object*> `ParentObject`: The parent object.<br>▪ <*System.Array*> `Elements`: Elements to add as children.<br>▪ <*Object*> `Follower`: The follower, elements will be inserted before this element, maybe null. | None |
| CreateDataObject | Creates an ICaDataObject as child for the given parent. | ▪ <ICaDataObjectType (refer to ICaDataObjectType <<Interface>> on page 154)> `CreatableType`: The type of the object which should be created<br>▪ <ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)> `Parent`: The ICaDataObject which should be the parent for the new created object. Default value is null. | The new created ICaDataObject.<br>▪ ICaDataObject (refer to ICaDataObject <<Interface>> on page 150) |
| ExportXmlData | prototype version of exporting. | ▪ <*String*> `filename`: The parameter filename. | None |
| FindByXPath | Finds and return elements of a provided relation matching a given XPath 1.0 expression. Currently only supported for CommunicationMatrix and Bus Configuration relations. | ▪ <*String*> `Pattern`: XPath expression compatible to the XPath 1.0 standard<br>▪ <*Object*> `StartObject`: For one object of typeICaDataObject (refer to ICaDataObject <<Interface>> on page 150): use this object as start element. Other types: Reserved for future use | Elements of the used relation matching the given XPath 1.0 expression<br>▪ ICaObjects (refer to ICaObjects <<Collection>> on page 158) |
| GetCreatableTypes | Gets the creatable types which are supported by an ICaDataObject. | ▪ <ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)> `ParentDataObject`: The ICaDataObject which should | Collection of creatable types, maybe empty<br>▪ ICaDataObjectTypes (refer to ICaDataObjectTypes <<Collection>> on page 155) |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| | | be the parent for the new created object. Default value is null. | |
| GetElements | Gets the elements for the related Object. | ▪ <*Object*> `DataObject`: The data object to get the related elements for. | Collection of ICaObjects which represents the related elements, maybe empty.<br>▪ ICaObjects (refer to ICaObjects <<Collection>> on page 158) |
| GetTopNodes | Gets the top nodes of a relation. | None | Collection of ICaObjects which represents the top nodes. Maybe empty.<br>▪ ICaObjects (refer to ICaObjects <<Collection>> on page 158) |
| RemoveElements | Removes the given elements from the given Objects children. | ▪ <*Object*> `ParentObject`: The parent object.<br>▪ <*System.Array*> `Elements`: Elements to remove from the children. | None |
| SetElements | Sets the given elements to specified relation for the given Object. Removes the previous elements for this relation before. Attention: This method may be not supported by all available RelationAccessors. | ▪ <*System.Array*> `Elements`: Elements to set the relation.<br>▪ <*Object*> `DataObject`: Object to set the related elements for. | None |

[1)] <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaRelations (refer to ICaRelations <<Collection>> on page 160)

## ICaRelations <<Collection>>

**Description**

Use this interface to get a ICaRelation (refer to ICaRelation <<Interface>> on page 158)for accessing DataObjects with a specific relation. The currently supported relations are listed below with a short description. Look at the more detailed examples written in Python, which are located in the ToolAutomation folder, which is a sub folder of the customers demo folder. The implementing automation object for this interface is only valid in its active application. Any access after closing the application is undefined and can cause unpredictable results.

| Relation Accessor | Usages |
|---|---|
| ChannelSets | This relation is useful for hardware assignment. Use the ChannelSets relation to get the channel sets which are designated for a function. E.g. if you got an accessor for ChannelSets via the RelationAccessors.Item("ChannelSets") you can query with RelationAccessor.GetElements(<SignalConditioningPartofFunctionBlockItem>) for a collection of appropriate requirements of this FunctionBlock. Attention: With the SetElements method then you can assign a specific ChannelSet. |
| AllChannelSets | In comparison to the ChannelSets relation this relation gets all channel sets even if an assignment would result in a conflict. |
| HardwareRequirements | This relation is useful for hardware assignment. Use the HardwareRequirements relation to get the requirements which are designated for a function. E.g. if you got an accessor for HardwareRequirements via the RelationAccessors.Item("HardwareRequirements") you can query with RelationAccessor.GetElements(<SignalConditioningPartofFunctionBlockItem>) for a collection of appropriate requirements of this FunctionBlock. Attention: The SetElements method is not supported for the HardwareRequirements relation. To use the HardwareRequirements relation successfully you have to assign a ChannelSet before. |
| ApplicableChannels | This relation is useful for hardware assignment. Use the ApplicableChannels relation to get the channels which fit a hardware requirement. E.g. if you got a hardware requirement for a function block use the ApplicableChannels relation accessor (RelationAccessors.Item("ApplicableChannels")) to query for the channels: RelationAccessor.GetElements(<HardwareRequirementItem>). Attention: The SetElements method is not supported for the ApplicableChannels relation. |
| AssignedChannels | This relation is useful for hardware assignment. Use the AssignedChannels relation to query for or to set channels regarding hardware requirements. E.g. if you got a hardware requirement you can query with the AssignedChannels relation accessor (RelationAccessors.Item("AssignedChannels")) for the assigned channels: RelationAccessor.GetElements(<HardwareRequirementItem>). If you got applicable channels for a hardware requirement you can assign one of them with the SetElements method: RelationAccessor.SetElements(<HardwareRequirementItem>, <ApplicableChannelItems>). |
| ApplicationConfiguration | This relation is useful for application and task modeling. Use the ApplicationConfiguration relation to get the executable application via RelationAccessor.GetTopNodes(), to query for configurable elements with RelationAccessor.GetElements(<ParentItem>) or to add elements to an application process or to a task with RelationAccessor.SetElements(<ParentItem>). Create new tasks or timer events by querying for the creatable type with RelationAccessor.GetCreatableTypes(<ParentItem>) and calling RelationAccessor.CreateDataObject(<CreatableTypeItem>, <ParentItem>). |
| ModelCommunication | This relation is useful for configuring the communication regarding models and application processes. Use the ModelCommunication relation to get or create communication packages and to add or remove models to or from communication packages. Get the standard communication package and previously created packages with RelationAccessor.GetTopNodes(). Create new communication packages by querying for the creatable type with RelationAccessor.GetCreatableTypes() and calling RelationAccessor.CreateDataObject(<CreatableTypeItem>). |
| BuildConfiguration | This relation is useful for configuring the build configuration. Use RelationAccessor.GetTopNodes() to get the executable application and to query then for global build settings and build configuration sets via RelationAccessor.GetElements(<ExecutableApplicationItem>). Configure the global build settings via the properties of the ICaDataObject. Create new build configuration sets by getting the creatable types with |

| Relation Accessor | Usages |
|---|---|
| | RelationAccessor.GetCreatableTypes(<ExecutableApplicationItem>) and calling RelationAccessor.CreateDataObject(<CreatableTypeItem>, <ParentItem>). Get application processes from build configuration sets with RelationAccessor.GetElements(<BuildConfigurationSetItem>) and assign it to other build configuration sets via RelationAccessor.SetElements(<ParentItem>, <ApplicationProcessItems>); |
| DeviceConnectors | This relation is useful for getting or creating device connectors and device pins. Use the RelationAccessor.GetElements(<DeviceBlockItem>) to get the related device connectors and pins (including unresolved elements). Use the RelationAccessor.GetCreatableTypes(<DeviceBlockItem>) and the RelationAccessor.CreateDataObject(<CreatableTypeItem>, <DeviceBlockItem>) to create a device connector or a device pin. To assign a pin to a device port use the ports "AssignedPins" property and set its value to the port object. To assign pins to a different device connector or a different device block use the RelationAccessors.AddElements(<ParentItem>, <PinItems>) method. It is even possible to assign device connectors to a different device block. |
| Links | This relation is useful for getting all links from a parent object. Currently supported are blocks. Use the RelationAccessor.GetElements(<BlockItem>) method to get all links from a block. |
| PropertyDataObjects | This relation is useful for getting the ICaDataObjects in the hierarchy of the PropertyGrid TreeView. Because this hierarchy often represents no parent - child relation it may be not possible to get the objects via ICaDataObject.Item or ICaDataObject.Parent. Therefore this relation supports the RelationAccessor.GetElements(<DataObjectItem>> method to get these data objects. |
| CommunicationMatricesByClusters | This relation is useful for getting Communication Matrix elements in a hierarchy ordered by clusters. This hierarchy represents a cluster based communication, which means every ECU is ordered under its Communication Cluster, regardless used bus system type. |
| CommunicationMatricesByEcus | This relation is useful for getting Communication Matrix elements in a hierarchy ordered by ECUs. This hierarchy represents an ECU based communication, which means every ECU is ordered by itself, containing all bus system types it represents, regardless of the Communication Cluster its connected with. |
| BusConfigurations | This relation is useful for getting the configured Communication Matrix elements for all Bus Configurations. |
| CommunicationMatricesByClustersWithProperties | The relation is useful for getting Communication Matrix elements, including their properties, in a hierarchy ordered by clusters via XPath. Attention: This relation decreases the performance. Access as few properties as possible via this relation. To access a high number of properties, use the PropertyDataObjects relation instead. You cannot use XPath expressions with the PropertyDataObjects relation. |
| CommunicationMatricesByEcusWithProperties | The relation is useful for getting Communication Matrix elements, including their properties, in a hierarchy ordered by ECUs via XPath. Attention: This relation decreases the performance. Access as few properties as possible via this relation. To access a high number of properties, use the PropertyDataObjects relation instead. You cannot use XPath expressions with the PropertyDataObjects relation. |
| BusConfigurationsWithProperties | The relation is useful for getting the configured Communication Matrix elements, including their properties, for all Bus Configurations via XPath. Attention: This relation decreases the performance. Access as few properties as possible via this relation. To access a high number of properties, use the PropertyDataObjects relation instead. You cannot use XPath expressions with the PropertyDataObjects relation. |
| AssignableProviders | This relation is useful for assigning provider blocks, e.g. blocks providing a master APU. Use the AssignableProviders relation to get the providers which can be assigned to a |

| Relation Accessor | Usages |
|---|---|
| | given provider request: RelationAccessor.GetElements(<ProviderRequest>). Attention: The SetElements method is not supported for the AssignableProviders relation. |
| ProviderRequests | This relation is useful for assigning provider blocks, e.g. blocks providing a master APU. Use the ProviderRequests relation to get the provider requests a function block has. You can query a function block via RelationAccessor.GetElements(< FunctionBlock>) for its contained requests. Attention: The SetElements method is not supported for the ProviderRequests relation. The ProviderRequests relation supports groups of requests, for example, used in ECU Interface Configuration function blocks. To get single requests from groups you have to call the relation a second time with the respective group element. |
| AssignedProviders | This relation is useful for assigning provider blocks, e.g. blocks providing a master APU. Use the AssignedProviders relation to query for or to set providers for a given block request. E.g. if you got a provider request for a function block you can query with the AssignedProviders relation accessor for the assigned providers: RelationAccessor.GetElements(< ProviderRequest >). If you got assignable providers for a provider request you can assign one of them with the SetElements method: RelationAccessor.SetElements(< ProviderRequest >, < AssignableProviderItems>). |
| FeatureProviders | This relation is useful for full configuration of provider blocks, e.g. blocks providing an I/O function trigger, or getting such contained feature providers for later assignment via the AssignedProviders relation. Use the FeatureProviders relation to get the feature providers of a function block. You can query a function block via RelationAccessor.GetElements(< FunctionBlock>) for its contained feature providers. Attention: The SetElements method is not supported for the FeatureProviders relation. |
| HardwareNetworkView | This relation is useful to get objects according the network view hierarchy, e.g. assigning provider blocks, e.g. displayed in the hardware browser when activating network view instead of assembly view. |

**Properties** The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| AllChannelSets | Gets the all channel sets relation. | Get | ICaRelation (refer to ICaRelation <<Interface>> on page 158) |
| ApplicableChannels | Gets the applicable channels relation. | Get | ICaRelation (refer to ICaRelation <<Interface>> on page 158) |
| ApplicationConfiguration | Gets the application configuration relation. | Get | ICaRelation (refer to ICaRelation <<Interface>> on page 158) |
| AssignableProviders | Gets the assignable providers relation. | Get | ICaRelation (refer to ICaRelation <<Interface>> on page 158) |
| AssignedChannels | Gets the assigned channels relation. | Get | ICaRelation (refer to ICaRelation <<Interface>> on page 158) |
| AssignedProviders | Gets the assigned providers relation. | Get | ICaRelation (refer to ICaRelation <<Interface>> on page 158) |

| Name | Description | Get/Set | Type |
|---|---|---|---|
| BuildConfiguration | Gets the build configuration relation. | Get | ICaRelation (refer to ICaRelation <<Interface>> on page 158) |
| BusConfigurations | Gets the bus configurations relation. | Get | ICaRelation (refer to ICaRelation <<Interface>> on page 158) |
| BusConfigurationsWithProperties | Gets the bus configurations with properties relation. | Get | ICaRelation (refer to ICaRelation <<Interface>> on page 158) |
| ChannelSets | Gets the channel set relation. | Get | ICaRelation (refer to ICaRelation <<Interface>> on page 158) |
| CommunicationMatricesByClusters | Gets the communication matrices by clusters relation. | Get | ICaRelation (refer to ICaRelation <<Interface>> on page 158) |
| CommunicationMatricesByClustersWithProperties | Gets the communication matrices by clusters with properties relation. | Get | ICaRelation (refer to ICaRelation <<Interface>> on page 158) |
| CommunicationMatricesByEcus | Gets the communication matrices by ecus relation. | Get | ICaRelation (refer to ICaRelation <<Interface>> on page 158) |
| CommunicationMatricesByEcusWithProperties | Gets the communication matrices by ecus with properties relation. | Get | ICaRelation (refer to ICaRelation <<Interface>> on page 158) |
| Count | Gets the number of available Relations | Get | *Signed 32 Bit Integer* |
| DeviceConnectors | Gets the device connectors relation. | Get | ICaRelation (refer to ICaRelation <<Interface>> on page 158) |
| FeatureProviders | Gets the feature providers relation. | Get | ICaRelation (refer to ICaRelation <<Interface>> on page 158) |
| HardwareNetworkView | Gets the hardware-network view relation. | Get | ICaRelation (refer to ICaRelation <<Interface>> on page 158) |
| HardwareRequirements | Gets the hardware requirements relation. | Get | ICaRelation (refer to ICaRelation <<Interface>> on page 158) |
| Links | Gets the links relation. | Get | ICaRelation (refer to ICaRelation <<Interface>> on page 158) |
| ModelCommunication | Gets the model communication relation. | Get | ICaRelation (refer to ICaRelation <<Interface>> on page 158) |
| PropertyDataObjects | Gets the property data objects relation. | Get | ICaRelation (refer to ICaRelation <<Interface>> on page 158) |

| Name | Description | Get/Set | Type |
|---|---|---|---|
| ProviderRequests | Gets the provider requests relation. | Get | ICaRelation (refer to ICaRelation <<Interface>> on page 158) |

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Contains | Determines whether an item with the specified accessor name is contained in the collection. | ▪ *<String>* `AccessorName`: Name of the accessor. | *true*if [contains] [the specified accessor name]; otherwise,*false*.<br>▪ *Boolean* |
| Item | Gets the item by the specified index. | ▪ *<String>* `Index`: Accessor name or a numeric index value. | Return value of the method.<br>▪ ICaRelation (refer to ICaRelation <<Interface>> on page 158) |

1) <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaActiveApplication (refer to ICaActiveApplication <<Interface>> on page 203)

## ICaStrings <<Collection>>

**Description**

Provides access to strings that provide information like component types, working paths, and so on.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| Count | Returns the number of strings. | Get | *Signed 32 Bit Integer* |

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Contains | Determines whether the specified string is contained in the collection. | ▪ *<String>* `item`: The item. | *true*if string is in the collection otherwise,*false*.<br>▪ *Boolean* |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Item | Returns a string according to the specified index. | ▪ *<Signed 32 Bit Integer>* `Index`: Numeric index value. | The found string.<br>▪ *String* |

1) <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaActiveApplication (refer to ICaActiveApplication <<Interface>> on page 203)
- ICaComponent (refer to ICaComponent <<Collection>> on page 141)
- ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)
- ICaWorkingViews (refer to ICaWorkingViews <<Collection>> on page 171)

## ICaTransaction <<Interface>>

**Description**

Represents a transaction.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| CanRollback | Indicates if it is possible to perform a rollback for the operation which was executed during this transaction. | Get | *Boolean* |
| IsInWriteState | Returns the current state of the ICaTransaction object. A read transaction returns always false. A write transaction returns true if it is in write state, otherwise false. | Get | *Boolean* |
| IsWriteTransaction | Indicates if this transaction is a write transaction. A read transaction returns always false. A write transaction returns always true. | Get | *Boolean* |

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Close | Closes the transaction. After the relation was closed every access to it will throw an exception. | None | None |
| Rollback | Performs a rollback for the operation which was executed during this transaction. | None | None |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| SwitchToReadState | Switches a write transaction into read state. If a transaction is in read state nothing will be done. | None | None |
| SwitchToWriteState | Switches a write transaction into write state. If a write transaction is in write state nothing will be done. If this operation is performed on a read transaction a not supported exception will be thrown. | None | None |

[1)] <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaTransactionCreator (refer to ICaTransactionCreator <<Interface>> on page 167)

## ICaTransactionCreator <<Interface>>

**Description**

This interface provides creating a transaction. A transaction can be used to encapsulate more than one automation operation.

**Properties**

The element has no properties.

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| CreateReadTransaction | Creates a read transaction. | None | ICaTransaction representing a read transaction.<br>▪ ICaTransaction (refer to ICaTransaction <<Interface>> on page 166) |
| CreateWriteTransaction | Creates a write transaction. | ▪ *<String>* `Description`: String describing the purpose of the action which should be wrapped by this transaction.<br>▪ *<Boolean>* `isInitiallyInReadState` | ICaTransaction representing a write transaction<br>▪ ICaTransaction (refer to ICaTransaction <<Interface>> on page 166) |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| | | : if set to*true*transaction is initially in read state. | |

[1)] <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaActiveApplication (refer to ICaActiveApplication <<Interface>> on page 203)

# ICaWorkingView <<Collection>>

**Description**

Provides access to the working view of an application. A working view contains a number of ports that can be added and removed by adding and removing ICaDataObjects. ICaDataObjects represent not only, single ports, but also blocks and subblocks (for example, Devices, IOFunctions), which themselves are a set of ports. Adding ICaDataObjects that represent blocks means adding only these ports. Iterating through the collection of ICaDataObjects of a working view means iterating through the collection of ports, not a collection of blocks or subblocks. An object which implements this interface is valid in the context of the current active application. If the corresponding working view is deleted, each access to the object can cause an exception. If the current application is closed, the object is no longer valid. Access is then undefined and can result in an exception.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| Count | Gets the number of data objects. The number of data objects refers to the ports in the working view. | Get | *Signed 32 Bit Integer* |
| FullName | Gets the full name of the working view. The string includes the working view group. | Get | *String* |
| ImplementingType | Returns the name of the interface that is implemented by this object. | Get | *String* |
| Name | Gets or sets the name of the working view. Forward slashes and backslashes are not allowed in a name. | Get/Set | *String* |
| WorkingViewGroup | Gets the working view group. The returned string can be empty if the working view is in the root level. | Get | *String* |

**Methods** The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Add | Adds a data object to the working view.<br>If data object to add is invalid the behavior is undefined. | ▪ <ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)> `DataObject`: Data object to add. | None |
| Clear | Removes all data objects. | None | None |
| Export | Exports the content of a working view to a file. If the file exist, it will be overwritten. Export is not supported by a global working view. | ▪ <*String*> `FullPath`: Full path to the file which should contain the working view content. | None |
| Import | Imports the content of a file to the working view. The content of the file will be imported due to the specified import mode. Supported import modes are: Replace, Merge and Add. Import is not supported by a global working view. | ▪ <*String*> `FullPath`: Full path to the file to import the content from.<br>▪ <ImportToWorkingViewMode (refer to ImportToWorkingViewMode <<Enumeration>> on page 182)> `ImportMode`: The import mode. | None |
| Item | Returns a data object according to the specified index or full path. Full path is currently not supported. | ▪ <*String*> `Index`: Full path of the data object or a numeric index value. | The found data object.<br>▪ ICaDataObject (refer to ICaDataObject <<Interface>> on page 150) |
| Remove | Removes the data object from the working view. It is not possible to remove data objects from the Global working view.<br>If data object to remove is invalid the behavior is undefined. | ▪ <ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)> `DataObject`: DataObject to remove. | None |
| ShowModelCommunication | Shows a model communication window for this working view. With Release 2017B the parameter NewWindow is no more supported. | ▪ <*Boolean*> `NewWindow`: Deprecated: This parameter will be ignored (always false). | None |
| ShowSignalChain | Shows a window for this working view. With Release 2017B the parameter NewWindow is no more supported. | ▪ <*Boolean*> `NewWindow`: Deprecated: This parameter will be ignored (always false). | None |

[1)] <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaWorkingViews (refer to ICaWorkingViews <<Collection>> on page 171)

## ICaWorkingViewGroup <<Interface>>

**Description**

Provides access to a WorkingViewGroup item. A working view group is represented by a string containing a path under which working views are located. Normally you can use these strings representing working view groups for work (e.g. Create, Copy,...) if you want to work with working views which are located under working view groups other than root. (The root is represented by an empty string and is not a valid group.) Therefore the main purpose of a WorkingViewGroupItem object is to add/remove it to/from the selection. If you need a WorkingViewGroup item for such an operation just call GetWorkingViewGroupItem(FullName) from the WorkingViews interface to get an object which represents a WorkingViewGroup working path string.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| FullName | Returns the full name (or working path) for the WorkingViewGroup item. | Get | *String* |
| ImplementingType | Returns the name of the interface that is implemented by this object. | Get | *String* |
| Name | Returns the name of the WorkingViewGroup item. | Get | *String* |
| Parent | Returns the parent WorkingViewGroup full name of this WorkingViewGroup item. Empty string for root. | Get | *String* |

**Methods**

The element has no methods.

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaWorkingViews (refer to ICaWorkingViews <<Collection>> on page 171)

# ICaWorkingViews <<Collection>>

**Description**

This interface is to access the working views and working view groups of an application. The working view elements are located under a working path which is called a working view group. There are always two permanent views: Global and Temporary. These views can never be removed. These are located in the root working view group (""). A working view is accessible via the ICaWorkingView interface and a working view group is represented by a string. A working view group itself can contain other working view groups that then represent a hierarchy (for example "View_Group_001\View_001" or "View_Group_002\ViewGroup_003\View_005"). An object that implements the ICaWorkingViews interface is valid in the context of the current active application. If this application is closed, the object is no longer valid even after reopening the application. Access to a invalid object is undefined and can cause unpredictable results.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| Count | Returns the number of working views.<br>The count includes all WorkingViews regardless of the WorkingViewGroup. | Get | *Signed 32 Bit Integer* |

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Add | Adds a new working view to the specified working view group. A working view group is created if it does not exist. Slashes or backslashes can be used to separate the WorkingViewGroups. | ▪ *<String>* `WorkingViewGroup`: The WorkingViewGroup under which the new WorkingView will be located. String.Empty for root.<br>▪ *<String>* `Name`: The Name of the WorkingView. String.Empty for default name. | The new WorkingView.<br>▪ ICaWorkingView (refer to ICaWorkingView <<Collection>> on page 168) |
| AddWorkingViewGroup | Adds a new working view group to the specified parent. If no name is specified, a default name for the working view group is created automatically.<br>Slashes and back slashes are separators and not allowed for the name. Use an empty string for the root directory. | ▪ *<String>* `Parent`: The full name of the parent WorkingViewGroup. String.Empty for root.<br>▪ *<String>* `Name`: The name of the new WorkingViewGroup. | The full name of the new WorkingViewGroup.<br>▪ *String* |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Clear | Removes all working view groups and working views except "Global" and "Temporary". | None | None |
| Contains | Tests if working view with the specified full name is contained in the collection. If the parameter "FullName" identifies a WorkingViewGroup, false will be returned. | ▪ <*String*> `FullName`: FullName of WorkingView searched for. | True if the collection contains a WorkingView with given FullName<br>▪ *Boolean* |
| Copy | Creates a copy of the current working view or working view group below the specified working view group. A working view group is created if it does not exist. The copy gets a new name if an item with the same name exists in the destination WorkingViewGroup. | ▪ <*String*> `FullName`: The full name.<br>▪ <*String*> `DestinationWorkingViewGroup`: The destination WorkingViewGroup. | FullName of the copy of the new WorkingView or WorkingViewGroup.<br>▪ *String* |
| GetParentWorkingViewGroup | Gets the parent working view group. | ▪ <*String*> `ChildFullName`: Full name of the child WorkingView or WorkingViewGroup. Must exist. | String with FullName of the WorkingViewGroup under which the child is located. Empty string for root.<br>▪ *String* |
| GetWorkingViewFullNamesFromParent | Gets the full names of all working views for the specified working view group. The returned strings collection doesn't include FullNames of WorkingViews which are located in WorkingViewGroups, which are present under the given ParentWorkingViewGroup. If no WorkingView can be found under the specified parent, an empty collection will be returned. The ICaStrings collection doesn't reflect changes. | ▪ <*String*> `ParentWorkingViewGroup` : The WorkingViewGroup to get WorkingViews from. Empty string for root. | Strings collection with FullNames.<br>▪ ICaStrings (refer to ICaStrings <<Collection>> on page 165) |
| GetWorkingViewGroupItem | Creates an object that represents the specified full name. If an empty string is used as argument ("root"), null is returned. | ▪ <*String*> `FullName`: FullName of existing WorkingViewGroup. Must exist. | WorkingViewGroup representing the given FullName or null for root.<br>▪ ICaWorkingViewGroup (refer to ICaWorkingViewGroup <<Interface>> on page 170) |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| GetWorkingViewGroups | Gets the collection of working view groups. An empty string representing the root working view group is always included in the returned collection. Currently no filter is supported and the call will always return all present WorkingViewGroups of the application. An empty string is returned for the root path. The ICaStrings collection doesn't reflect changes made by deleting or adding WorkingViewGroups after getting it from the ICaWorkingViews object. | ▪ *<String>* `Filter`: The filter. Default value is null for all WorkingViewGroups. | Collection of strings with the WorkingViewGroups.<br>▪ ICaStrings (refer to ICaStrings <<Collection>> on page 165) |
| GetWorkingViewGroupsFromParent | Gets the working view groups from a parent working view group.<br>The returned strings collection doesn't include FullNames of WorkingViewGroups in "sub" WorkingViewGroups, which are located under the given ParentWorkingViewGroup. If no WorkingViewGroup is present under the specified parent, an empty collection will be returned. The ICaStrings collection doesn't reflect changes. | ▪ *<String>* `ParentWorkingViewGroup`: The parent WorkingViewGroup. Empty string for root. | A collection of strings containing the full names of the WorkingViewGroups.<br>▪ ICaStrings (refer to ICaStrings <<Collection>> on page 165) |
| Item | Returns a working view according to the specified index or full name.<br>If the FullName is used as parameter slashes or backslashes can be used to separate the WorkingViewGroups. | ▪ *<String>* `Index`: FullName of the WorkingView or a numeric index value. | The found WorkingView object.<br>▪ ICaWorkingView (refer to ICaWorkingView <<Collection>> on page 168) |
| Move | Moves the working view or working view group to the specified working view group. A working view group is created if it does not exist. | ▪ *<String>* `FullName`: Full name of the WorkingView or WorkingViewGroup which should be moved.<br>▪ *<String>* `NewParentWorkingViewGroup`: The new WorkingViewGroup parent. | None |
| Remove | Removes the specified working view.<br>If parameter "WorkingView" refers to the global or | ▪ <ICaWorkingView (refer to ICaWorkingView <<Collection>> on page 168)> `WorkingView`: | None |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| | temporary WorkingView no action will be taken. | The working view to remove. | |
| RemoveByFullName | Removes the specified working view or working view group. If the given parameter identifies a WorkingViewGroup all WorkingViews and WorkingViewGrooups that are located in that path will be removed. The root cannot be removed. Use Clear() to remove all WorkingViews and WorkingViewGroups except Global and Temporary view from the application. | ▪ *<String>* `FullName`: The full name of the item to remove. | None |

1) <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaActiveApplication (refer to ICaActiveApplication <<Interface>> on page 203)

## MatchingPlatformConnectionState <<Enumeration>>

**Description**

The state of indicating if a matching platform is connected.

**Enumeration values**

The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| NotConnected | No or no matching platform is connected. | 0 |
| Connected | A matching platform is connected. | 1 |

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaActiveApplication (refer to ICaActiveApplication <<Interface>> on page 203)

## ModelTopologyCreateMode <<Enumeration>>

**Description**

The mode how to create a model topology. The file format changed for ConfigurationDesk 4.3 from *.mtf to *.mtfx.

**Enumeration values**

The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| FileImportMdl | Creates a model topology by importing a Simulink *.mdl file. | 1 |
| FileImportMtf | Creates a model topology by importing a *.mtfx file. | 2 |
| FileImportMcd | Creates a model topology by importing a *.mcd file. | 3 |
| EmptyTopology | Creates an empty model topology without any model. | 4 |
| FileImportSlx | Creates a model topology by importing an *.slx file. | 5 |
| FileImportOther | Creates a model topology by importing a file with none of the above extensions. | 6 |

# Enumeration Properties

**Where to go from here**

Information in this section

## AveragingLevel <<Enumeration>>

**Description** Definition of the Averaging Level type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| Precise | Represents the precise averaging level type. | 1 |
| Dynamic | Represents the dynamic averaging level type. | 2 |
| DefinedByModel | Represents the defined by model averaging level type. | 3 |

## BitOrder <<Enumeration>>

**Description** Definition of the BitOrder type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| Inverse | Represents the inverse bitorder type. | 0 |
| Normal | Represents the normal bitorder type. | 1 |

## BlockColor <<Enumeration>>

**Description** Definition of the block colors.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| LightBlue | Represents the color light blue (default). | 0 |
| White | Represents the color white. | 1 |
| Red | Represents the color red. | 2 |
| Green | Represents the color green. | 3 |
| Blue | Represents the color blue. | 4 |
| Cyan | Represents the color cyan. | 5 |

| Name | Description | Value |
|---|---|---|
| Magenta | Represents the color magenta. | 6 |
| Yellow | Represents the color yellow. | 7 |
| Gray | Represents the color gray. | 8 |
| Orange | Represents the color orange. | 9 |
| DarkGreen | Represents the color dark green. | 10 |
| Beige | Represents the color beige. | 11 |

## ChannelType <<Enumeration>>

**Description** Definition of the ChannelType type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| FlexibleOut1 | Represents the flexibleout1 channel type. | 0 |
| DigitalOut1 | Represents the digitalout1 channel type. | 1 |
| AnalogOut1 | Represents the analogout1 channel type. | 2 |
| AnalogOut4 | Represents the analogout4 channel type. | 3 |
| AnalogOut3 | Represents the analogout3 channel type. | 4 |
| ResistanceOut1 | Represents the resistanceout1 channel type. | 5 |
| FlexibleIn1 | Represents the flexiblein1 channel type. | 6 |
| DigitalIn1 | Represents the digitalin1 channel type. | 7 |
| AnalogIn1 | Represents the analogin1 channel type. | 8 |
| FlexibleIn2 | Represents the flexiblein2 channel type. | 9 |
| Bus1 | Represents the bus1 channel type. | 10 |
| CAN1 | Represents the can1 channel type. | 11 |
| Lin1 | Represents the lin1 channel type. | 12 |
| FlexRay1 | Represents the flexray channel type. | 13 |
| PowerSwitch1 | Represents the powerswitch1 channel type. | 14 |
| PowerSwitch2 | Represents the powerswitch2 channel type. | 15 |
| PowerControl1 | Represents the powercontrol1 channel type. | 16 |
| AnalogOut2 | Represents the analogout2 channel type. | 17 |
| AnalogIn2 | Represents the analogin2 channel type. | 18 |
| Load1 | Represents the load1 channel type. | 19 |

## CylinderStates <<Enumeration>>

**Description** Definition of the CylinderStates type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| Inactive | Represents the inactive cylinder state. | 0 |
| Active | Represents the active cylinder state. | 1 |

## DigitalOutputMode <<Enumeration>>

**Description** Definition of the DigitalOutputMode type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| Unspecified | Represents the Unspecified digital output mode. | -1 |
| Undefined | Represents the Undefined digital output mode. | 0 |
| Switch | Represents the switch digital output mode. | 1 |
| LowSideSwitch | Represents the lowside switch digital output mode. | 2 |
| TriState | Represents the TriState digital output mode. | 3 |
| HighSideSwitch | Represents the highside switch digital output mode. | 4 |
| PushPull | Represents the push pull digital output mode. | 8 |

## Direction <<Enumeration>>

**Description** Definition of the Direction type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| In | Represents the In direction type. | 1 |
| Out | Represents the Out direction type. | 2 |

| Name | Description | Value |
|---|---|---|
| Reference | Represents the Reference direction type. | 4 |
| BiDirectional | Represents the BiDirectional direction type. | 8 |

## EdgeType <<Enumeration>>

**Description** Definition of the EdgeType type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| None | Represents the none edge type type. | 0 |
| Rising | Represents the rising edge type type. | 1 |
| Falling | Represents the falling edge type type. | 2 |
| Both | Represents the both edge type type. | 3 |
| BothRisingFirst | Represents the both rising first edge type type. | 4 |
| BothFallingFirst | Represents the both falling first edge type type. | 5 |

## EncoderType <<Enumeration>>

**Description** Definition of the EncoderType type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| Rotary | Represents the rotary encoder type. | 1 |
| Linear | Represents the linear encoder type. | 2 |

## EventTriggerCondition <<Enumeration>>

**Description** Definition of the EventTriggerCondition type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| NoEvent | Represents the no event event trigger condition. | 0 |
| FirstEdge | Represents the first edge event trigger condition. | 1 |
| WindowEnding | Represents the window ending event trigger condition. | 2 |

## ExecutionMode <<Enumeration>>

**Description** Definition of the ExecutionMode type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| StateDependent | Represents the state dependent execution mode. | 1 |
| Immediate | Represents the immediate execution mode. | 2 |

## FunctionMode <<Enumeration>>

**Description** Definition of the FunctionMode type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| PwmMode | Represents the pwm mode function mode. | 1 |
| FrequencyMode | Represents the frequency mode function mode. | 2 |

## HighSideReference <<Enumeration>>

**Description** Definition of the HighSideReference type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| Individual | Represents the individual reference highside reference. | 1 |
| Vbat | Represents the external reference highside reference. | 2 |
| Unused | Represents the unused highside reference. | 3 |
| Shared | Represents the shared highside reference. | 4 |
| Shared2 | Represents the shared2 highside reference. | 5 |

## IdleValue <<Enumeration>>

**Description** Definition of the IdleValue type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| FirstValue | Represents the firstvalue idlevalue type. | 1 |
| LastValue | Represents the lastvalue idlevalue type. | 2 |
| OtherValue | Represents the othervalue idlevalue type. | 3 |

## ImportToWorkingViewMode <<Enumeration>>

**Description** Definition of the ImportToWorkingViewMode

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| Add | Specific elements will be added as duplicates, even if entities with the same ID have been found. All elements that were present before the operation stay in the application. | 0 |

| Name | Description | Value |
|---|---|---|
| Merge | All elements that can be found during import are reused. All elements that were present before the operation stay in the application. | 1 |
| Replace | All elements in context are removed before the import operation. | 2 |

## InitializationMode <<Enumeration>>

**Description** Definition of the InitializationMode type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| First | Represents the first simulation initialization mode. | 1 |
| Each | Represents the every simulation initialization mode. | 2 |

## JitterAndLatencyOptimization <<Enumeration>>

**Description** Lets you specify the jitter and latency run-time behavior of the task. To configure a task as 'NoJitterLowLatency', it must be the only task in the application process and triggered by a timer event. When you use custom code (e.g., custom I/O functions) and select 'NoJitterLowLatency', there might be functionality issues: - The background task of the real-time application is not executed. - Tasks with â€˜No jitter, low latencyâ€™ cannot be used with third-party I/O in the same application process. It is not recommended to use system calls within a â€˜NoJitterLowLatencyâ€™ task, because they re-introduce jitter.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| Standard | Represents the standard task mode. | 0 |
| LowJitterLowLatency | Represents the low jitter, low latency optimization mode. | 1 |
| NoJitterLowLatency | Represents the no jitter, low latency optimization mode. | 2 |

## LoadManualChecking <<Enumeration>>

**Description** Definition of the LoadManualChecking type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| BySystem | Represents the by system load manual checking. | 1 |
| ByUser | Represents the by user load manual checking. | 2 |

## MappingType <<Enumeration>>

**Description** The type of a mapping relation.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| None | No mapping relation. | 0 |
| DeviceMapping | Device mapping relation. | 1 |
| ModelMapping | Model mapping relation. | 2 |
| All | Considers all mapping types. | 3 |

## MeasurementMode <<Enumeration>>

**Description** Definition of the MeasurementMode type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| FreeRunning | Represents the free running measurement mode. | 1 |
| SingleTriggered | Represents the single triggered measurement mode. | 2 |

## MeasurementMode2 <<Enumeration>>

**Description** Definition of the MeasurementMode2 type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| DutyCycle | Represents the duty cycle measurementmode2 type. | 4 |
| Frequency | Represents the frequency measurementmode2 type. | 8 |
| Both | Represents the both measurementmode2 type. | 16 |

## MeasurementPoint <<Enumeration>>

**Description** Definition of the MeasurementPoint type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| Load | Represents the load measurement point. | 1 |
| Ecu | Represents the ECU measurement point. | 2 |

## OvercurrentProtection <<Enumeration>>

**Description** Definition of the OvercurrentProtection type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| Saturation | Saturation | 0 |
| Shutdown | Shutdown | 1 |

## PhaseUpdateMode <<Enumeration>>

**Description** Definition of the PhaseUpdateMode type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| Immediate | Represents the immediate phase update mode. | 1 |
| Smoothed | Represents the smoothed phase update mode. | 2 |

## Polarity <<Enumeration>>

**Description** Definition of the Polarity type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| ActiveLow | Represents the active low polarity type. | 0 |
| ActiveHigh | Represents the active high polarity type. | 1 |

## Potential <<Enumeration>>

**Description** Definition of the Potential type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| Unbound | Represents the Unbound potential type. | 1 |
| Bound | Represents the Bound potential type. | 2 |
| Ground | Represents the Ground potential type. | 3 |
| VBat | Represents the VBat potential type. | 4 |
| Unused | Represents the Unbound potential type. | 5 |

## ReadMode <<Enumeration>>

**Description** Definition of the ReadMode type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| Contiguous | Represents the contiguous read mode. | 1 |
| Overlapped | Represents the overlapped read mode. | 2 |

## Role <<Enumeration>>

**Description** Definition of the Role type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| Signal | Represents the Signal role type. | 1 |
| HighReference | Represents the HighReference role type. | 2 |
| LowReference | Represents the LowReference role type. | 3 |
| LoadSignal | Represents the LoadSignal role type. | 4 |
| LoadReference | Represents the LoadReference role type. | 5 |
| SignalInternal | Represents the SignalInternal role type. | 6 |
| LowReferenceInternal | Represents the LowReferenceInternal role type. | 7 |
| Unused | Represents the Unused role type. | 8 |

## SensorMode <<Enumeration>>

**Description** Definition of the SensorMode type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| Normal | Represents the normal sensor mode. | 1 |
| Reverse | Represents the reverse sensor mode. | 2 |

## SignalMode <<Enumeration>>

**Description** Definition of the SignalMode type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| Current | Represents the current signal mode. | 1 |
| Voltage | Represents the voltage signal mode. | 2 |

## StandstillBehavior <<Enumeration>>

**Description** Definition of the StandstillBehavior type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| PitchSynchronousTransmission | Represents the pitch synchronous transmission stand still behavior. | 1 |
| PeriodicTransmission | Represents the periodic transmission stand still behavior. | 2 |

## Termination <<Enumeration>>

**Description** Definition of the Termination type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| On | Represents the on termination type. | 0 |
| Off | Represents the off termination type. | 1 |
| R1600Ohm | Represents the r1600ohm termination type. | 2 |
| R10000Ohm | Represents the r10000ohm termination type. | 3 |

## TransceiverType <<Enumeration>>

**Description** Definition of the TransceiverType type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| Iso9141Lin | Represents the iso9141lin transceiver type. | 0 |
| Iso118982HighspeedCan | Represents the iso118982highspeedcan transceiver type. | 1 |
| Iso118983FaulttolerantCan | Represents the iso118983faulttolerantcan transceiver type. | 2 |
| PiggybackModule | Represents the piggybackmodule transceiver type. | 3 |

## TransferType <<Enumeration>>

**Description** Type of settings transfer.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| All | Transfers all settings. | 0 |
| AllowedFailureClasses | Transfers only the allowed failure classes. | 1 |
| LoadRejection | Transfers only the load rejection setting. | 2 |

## TriggerEdgeType <<Enumeration>>

**Description** Definition of the TriggerEdgeType type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| Rising | Represents the rising trigger edge type. | 1 |
| Falling | Represents the falling trigger edge type. | 2 |
| BothRisingFirst | Represents the both rising first trigger edge type. | 3 |
| BothFallingFirst | Represents the both falling first trigger edge type. | 4 |

## UpdateMode <<Enumeration>>

**Description** Definition of the UpdateMode type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| Synchronous | Represents the synchronous update mode. | 1 |
| Asynchronous | Represents the asynchronous update mode. | 2 |

## VoltagePolarity <<Enumeration>>

**Description** Definition of the Voltage Polarity type.

**Enumeration values** The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| Positive | Represents the positive voltage polarity type. | 0 |
| Negative | Represents the negative voltage polarity type. | 1 |

# Framework

**Where to go from here**

Information in this section



## ControlbarTabsLayout <<Enumeration>>

**Description**

Types of controlbar tabs layout.

**Enumeration values**

The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| AutoSized | Each controlbar tab contains the symbol and the name of the component that it represents. | 0 |
| Compressed | Only the active controlbar tab contains the symbol and name of the component that it represents. | 1 |
| SizeToFit | Each controlbar tab contains the symbol and the name of the component that it represents. | 2 |

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaMainWindow (refer to ICaMainWindow <<Interface>> on page 195)

# ICaApplicationMain <<Interface>>

**Description**

Provides access to the ConfigurationDesk application object. This is the main object to access, open or create projects and applications.

This is the only creatable object of the ConfigurationDesk tool automation.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| ActiveApplication | Returns the currently active application.<br>If no active application is available, null is returned. | Get | ICaActiveApplication (refer to ICaActiveApplication <<Interface>> on page 203) |
| ActiveProject | Returns the currently active project.<br>If no active project is available, null is returned. | Get | ICaActiveProject (refer to ICaActiveProject <<Interface>> on page 206) |
| ActiveProjectRoot | Returns the active project root folder.<br>If no project root is set, null is returned. | Get | ICaProjectRoot (refer to ICaProjectRoot <<Interface>> on page 213) |
| CommandLineArguments | Returns the application's command line arguments, including the full path of the application. | Get | *System.String[]* |
| Interpreter | Returns the internal Python interpreter. | Get | *dSPACE.PythonIDE.Automation.IPiInterpreter* |
| MainWindow | Gets the main window. | Get | ICaMainWindow (refer to ICaMainWindow <<Interface>> on page 195) |
| MessageDispatcher | Gets the message dispatcher. | Get | ICaMessageDispatcher (refer to ICaMessageDispatcher <<Interface>> on page 196) |
| Name | Returns the name of the application. | Get | *String* |
| PlatformManagement | Returns the platform management. | Get | *dSPACE.PlatformManagement.Automation.IPmPlatformManagement* |
| ProjectManagement | Gets the project management | Get | ICaProjectManagement (refer to ICaProjectManagement <<Interface>> on page 213) |
| ProjectRoots | Returns the collection of project root folders.<br>Returned collection may be empty. | Get | ICaProjectRoots (refer to ICaProjectRoots |

| Name | Description | Get/Set | Type |
|---|---|---|---|
| | | | <<Collection>> on page 214) |
| Projects | Returns a collection of projects in the active project root folder.<br>If no project root is set, null is returned.<br>Returned collection may be empty. | Get | ICaProjects (refer to ICaProjects <<Collection>> on page 215) |
| UserFunctions | Returns the user-defined functions.<br>The collection of the functions may be empty. | Get | ICaUserFunctions (refer to ICaUserFunctions <<Collection>> on page 200) |
| Version | Returns the current version of the application. | Get | *String* |

## Methods

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| GetCustomInformation | Gets the information which is specified by the collection parameter infos. If the information is not supported an exception will be generated. If information does not exist, empty array will be returned. If Infos collection is null or empty the queriable information types will be returned. Since Release 2017B it is no more possible to get the ExtendSignalChainOptions information.<br>Refer to Details on GetCustomInformation on page 194. | ▪ *<System.String[]>* `Infos`: Collection of strings which specifies the queried information. | Array with the information.<br>▪ *System.Array* |
| OpenApplication | Opens the project specified by the full path and activates the application with the specified name.<br>An already open project will be closed without saving it. | ▪ *<String>* `ProjectPath`: Full path name to the projects CDP file.<br>▪ *<String>* `ApplicationName`: Applications display name. | The Application opened, which is now active.<br>▪ ICaActiveApplication (refer to ICaActiveApplication <<Interface>> on page 203) |
| OpenProject | Opens the project specified by the full path.<br>Full path name should include the '*.cdp' project file extension. The projects location should be under a valid project root. An already open project will be closed without saving it. | ▪ *<String>* `ProjectPath`: Full path name to the projects CDP file. | The project opened, which is now active.<br>▪ ICaActiveProject (refer to ICaActiveProject <<Interface>> on page 206) |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Quit | Quits the application. | ▪ *<Boolean>* `SaveChanges`: Determines if to save changes. | None |
| SetCustomInformation | Sets the custom information for the information which is specified by the Infos collection. If information type is not supported, exception will be generated. Since Release 2017B it is no more possible to set the ExtendSignalChainOptions. Refer to Details on SetCustomInformation on page 194. | ▪ *<System.String[]>* `Infos`: The information for which to set the values.<br>▪ *<System.Array>* `Params`: The values to set for the information. | Array with possible results, otherwise empty array.<br>▪ *System.Array* |

[1)] <Type> `Name`: Description

### Details on GetCustomInformation

| Custom Information | Supported Information |
|---|---|
| CustomFunctionDirectory | Query for the path to the directory with custom functions and returns a string containing the path. Example usage: Application.GetCustomInformation(["CustomFunctionDirectory"]) |

### Details on SetCustomInformation

| Custom Information | Supported Information |
|---|---|
| CustomFunctionDirectory | Set the path to the directory with custom functions Example usage (Python): Application.SetCustomInformation(["CustomFunctionDirectory"], [r"C:\temp"]) |
| ImportCustomFunctions | Imports an archive file with custom functions from the given path either to the custom function folder or to the currently active project with optional parameter True. If active project is used an active application must be open. Example usage (Python): Application.SetCustomInformation(["ImportCustomFunctions"], [r"C:\IOConfigurationStuff\CustomFunctions.zip", True]) |
| PrecompileFMU | Precompiles an FMU, for example, to protect intellectual property. Possible usage with required path to the FMU (string), optional path and (optional) name of the precompiled file (string, default is same path and suffix '_precompiled'), optional compiler options (string), optional boolean flag to keep the sources (default is false) and optional target platform (string, default ist SCALEXIO). Example usage (Python): Application.SetCustomInformation(["PrecompileFMU"], [r"C:\Models\MyFMU.fmu"]) |
| PrecompileSIC | Precompiles a SIC, for example, to protect intellectual property. Possible usage with required path to the SIC (string), optional path and (optional) name of the precompiled file (string, default is same path and suffix '_precompiled'), optional compiler options (string), optional boolean flag to keep the sources (default is false) and optional target platform (string, default ist SCALEXIO). Example usage (python): |

| Custom Information | Supported Information |
|---|---|
| | Application.SetCustomInformation(["PrecompileSIC"], [r"C:\Models\MySIC.sic", None, None, True]) |
| PrecompileBSC | Precompiles a BSC, for example, to protect intellectual property. Possible usage with required path to the BSC (string), optional path and (optional) name of the precompiled file (string, default is same path and suffix '_precompiled'), optional compiler options (string), optional boolean flag to keep the sources (default is false) and optional target platform (string, default ist SCALEXIO). Example usage (python): Application.SetCustomInformation(["PrecompileBSC"], [r"C:\Models\MyBSC.bsc"]) |

**Event Interfaces**

The element provides the following event interfaces:

- ICaApplicationEvents (refer to ICaApplicationEvents <<EventInterface>> on page 217)

# ICaMainWindow <<Interface>>

**Description**

Provides access to the application's main window.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| AnimateAutoHiding | Enables or disables animation of windows auto hiding. | Get/Set | *Boolean* |
| Caption | Gets the main window's caption. | Get | *String* |
| ControlbarTabsLayout | Specifies the layout of the controlbar tabs. Use (0) for auto-sized, (1) for compressed or (2) for sized to fit. | Get/Set | ControlbarTabsLayout (refer to ControlbarTabsLayout <<Enumeration>> on page 191) |
| FullScreenModeEnabled | Enables or disables full screen mode. | Get/Set | *Boolean* |
| Height | Returns or sets the height of the main window. | Get/Set | *Signed 32 Bit Integer* |
| LargeToolbarIconsEnabled | Displays the toolbar buttons in large format. | Get/Set | *Boolean* |
| Left | Returns or sets the left position of the main window. | Get/Set | *Signed 32 Bit Integer* |
| ShortcutKeysVisible | If you select this option, the tool tip of a selected command contains information on its shortcut key, if available. | Get/Set | *Boolean* |
| State | Returns or sets the state of the main window. Use (0) to minimize, (1) to maximize, or (2) to restore the main window. | Get/Set | MainWindowState (refer to MainWindowState <<Enumeration>> on page 201) |

| Name | Description | Get/Set | Type |
|---|---|---|---|
| Top | Returns or sets the top position of the main window. | Get/Set | *Signed 32 Bit Integer* |
| Visible | Returns or sets the visibility of the main window. | Get/Set | *Boolean* |
| Width | Returns or sets the width of the main window. | Get/Set | *Signed 32 Bit Integer* |
| WorkbookModeEnabled | Enables or disables workbook mode. | Get/Set | *Boolean* |
| WorkbookTabPosition | Specifies whether to display the tabs at the top (0) or the bottom (1) of ConfigurationDesk's working area. | Get/Set | WorkbookTabPosition (refer to WorkbookTabPosition <<Enumeration>> on page 201) |

**Methods**

The element has no methods.

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaApplicationMain (refer to ICaApplicationMain <<Interface>> on page 192)

## ICaMessageDispatcher <<Interface>>

**Description**

Provides access to the message dispatcher to send messages to the Message Viewer or dSPACE Log.

**Properties**

The element has no properties.

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| SubmitErrorLogFile | Writes an error message in the ConfigurationDesk dSPACE Log | ▪ <*String*> `Message`: String message, must not be null or empty. | None |
| SubmitErrorLogView | Writes an error message in the ConfigurationDesk Message Viewer | ▪ <*String*> `Message`: String message, must not be null. | None |
| SubmitInfoLogFile | Writes an info message in the ConfigurationDesk dSPACE Log. | ▪ <*String*> `Message`: String message, must not be null or empty. | None |
| SubmitInfoLogView | Writes an info message in the ConfigurationDesk Message Viewer. | ▪ <*String*> `Message`: String message, must not be null or empty. | None |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| SubmitMessage | Submits the message. Currently, only the default type to set an info message in the dSPACE Log is supported. | ▪ *<String>* `Message`: The message, must not be null or empty.<br>▪ *<String>* `Type`: The type. Default is Info dSPACE Log with null or empty string. | None |
| SubmitWarningLogFile | Writes a warning message in the ConfigurationDesk dSPACE Log | ▪ *<String>* `Message`: String message, must not be null or empty. | None |
| SubmitWarningLogView | Writes a warning message in the ConfigurationDesk Message Viewer | ▪ *<String>* `Message`: String message, must not be null or empty. | None |

1) <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaApplicationMain (refer to ICaApplicationMain <<Interface>> on page 192)

## ICaProperties <<Collection>>

**Description**

Provides access to a collection of properties.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| Count | Returns the number of properties. | Get | *Signed 32 Bit Integer* |

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Contains | Tests if the property with the specified name is contained in the collection. | ▪ *<String>* `Name`: Name of property searched for. | True if the collection contains a property with given name.<br>▪ *Boolean* |
| Item | Returns a property according to the specified index or name. | ▪ *<String>* `Index`: Name of the property or a numeric index value. | The found property object.<br>▪ ICaProperty (refer to ICaProperty <<Interface>> on page 198) |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| TryGetItem | Tries to get a property item from the properties collection. | ▪ *<Object>* `Index`: Name or numerical index of item. | ICaProperty item if item is present in the collection, otherwise null.<br>▪ ICaProperty (refer to ICaProperty <<Interface>> on page 198) |

1) <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaBuildManagement (refer to ICaBuildManagement <<Interface>> on page 130)
- ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)

# ICaProperty <<Interface>>

**Description**

Provides methods to get or set a property value for a certain object.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| DisplayName | Gets the display name of this property object. The display name is the name shown in the PropertyGrid of ConfigurationDesk and may change in future. Therefore scripting with names should only rely on the automation name. | Get | *String* |
| IsCurrentlySignificant | Gets a value indicating whether this property is currently significant, i.e., if its setting is relevant in the current context or not. This may depend, for example, on the setting of a different property. | Get | *Boolean* |
| IsReadOnly | Gets a value indicating whether this property is read only. | Get | *Boolean* |
| Name | Gets the property name. | Get | *String* |
| Type | Gets the property value type. | Get | *String* |
| Value | Gets or sets the value of the property. | Get/Set | *Object* |

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| TrySetValue | Tries to the set the value of the property. | ▪ <*Object*> `Value`: The value to set. | True if value was set, otherwise false.<br>▪ *Boolean* |

[1)] <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaProperties (refer to ICaProperties <<Collection>> on page 197)

## ICaUserFunction <<Interface>>

**Description**

Provides access to a user-defined function.

A user-defined function is an external command that can be executed via ConfigurationDesk's 'User Functions' toolbar.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| Arguments | Returns or sets the arguments of the command of the user-defined function. | Get/Set | *String* |
| CaptureOutput | Enables or disables the capturing of output during execution. | Get/Set | *Boolean* |
| Command | Returns/Sets the command to be executed. | Get/Set | *String* |
| Description | Returns or sets the description of the user-defined function. | Get/Set | *String* |
| InitialDirectory | Returns or sets the initial folder where the command is executed. | Get/Set | *String* |
| Name | Returns or sets the name of the user-defined function. | Get/Set | *String* |
| ShowWindow | Shows or hides the commands window of the user-defined functions during execution. | Get/Set | *Boolean* |

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Execute | Executes the command of the user-defined function. | None | None |
| Remove | Removes the user-defined function from the collection. | None | None |
| SetImage | Defines the image of the user-defined function.<br>The image may be defined by a bitmap (BMP) or PNG file.<br>The image should be 16 pixels width and height. | ▪ *<String>* `ImageFileName`: Full path name of the image file. | None |

1) <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaUserFunctions (refer to ICaUserFunctions <<Collection>> on page 200)

# ICaUserFunctions <<Collection>>

**Description**

Provides access to the collection with user-defined functions of the application.

User-defined functions are external commands that can be executed via ConfigurationDesks 'User Functions' toolbar.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| Count | Returns the number of user-defined functions. | Get | *Signed 32 Bit Integer* |

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Add | Adds a new user-defined function to the collection. | ▪ *<String>* `Name`: Name of new user function. | The new added user function.<br>▪ ICaUserFunction (refer to ICaUserFunction <<Interface>> on page 199) |
| Contains | Tests if the user-defined function with the specified name is contained in the collection. | ▪ *<String>* `Name`: Name of user function searched for. | True if the collection contains a user function with given name.<br>▪ *Boolean* |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Item | Returns the user-defined function by index or name. | ▪ *<String>* `Index`: Name of the user function or a numeric index value. | The found user function object.<br>▪ ICaUserFunction (refer to ICaUserFunction <<Interface>> on page 199) |

1) <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaApplicationMain (refer to ICaApplicationMain <<Interface>> on page 192)

## MainWindowState <<Enumeration>>

**Description**

States of the application's main window.

**Enumeration values**

The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| Minimized | The main window is minimized. | 0 |
| Maximized | The main window is maximized. | 1 |
| Restored | The main window is restored. | 2 |

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaMainWindow (refer to ICaMainWindow <<Interface>> on page 195)

## WorkbookTabPosition <<Enumeration>>

**Description**

Types of workbook tab position.

**Enumeration values**

The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| Top | The workbook tab is on the top. | 0 |
| Bottom | The workbook tab is on the bottom. | 1 |

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaMainWindow (refer to ICaMainWindow <<Interface>> on page 195)

# Project and Application Management

**Where to go from here**

Information in this section



## FileType <<Enumeration>>

**Description**

The type of a file.

**Enumeration values**

The enumeration has the following values:

| Name | Description | Value |
|---|---|---|
| Unknown | A unknown file. | 0 |
| DeviceTopology | A DeviceTopology file. | 1 |
| Text | A text file. | 2 |
| Log | A logging file. | 3 |

| Name | Description | Value |
|---|---|---|
| ExternalCableHarness | A ExternalCableHarness file. | 4 |
| HardwareTopology | A HardwareTopology file. | 5 |
| ModelTopology | A ModelTopology file. | 6 |
| Script | A Python script file. | 7 |

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaFile (refer to ICaFile <<Interface>> on page 210)

## ICaActiveApplication <<Interface>>

**Description**

Provides access to the currently active application and is the basis for carrying out tasks in ConfigurationDesk. The active application interface provides access to certain components that can be used to configure a logical signal chain and/or to configure a hardware system. It provides functionality to build and download a real-time application. An active application automation object is only valid until closing it. Using this object after closing the application can cause unpredictable behavior

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| Algorithms | Returns an object that can be used to execute algorithms. | Get | ICaAlgorithms (refer to ICaAlgorithms <<Interface>> on page 135) |
| Application | Gets the application interface of this application. | Get | ICaApplication (refer to ICaApplication <<Interface>> on page 208) |
| BuildManagement | Returns the build management for building and downloading real-time applications. | Get | ICaBuildManagement (refer to ICaBuildManagement <<Interface>> on page 130) |
| Components | Returns the components of the application. Each component is contained only once in an application. | Get | ICaComponents (refer to ICaComponents <<Interface>> on page 149) |
| ComponentTypes | Gets the collection of supported component types. | Get | ICaStrings (refer to ICaStrings <<Collection>> on page 165) |

| Name | Description | Get/Set | Type |
|---|---|---|---|
| Description | Returns or sets the application's description. | Get/Set | *String* |
| Files | Returns the files managed by the application. Currently not supported. Collection is always empty. | Get | ICaFiles (refer to ICaFiles <<Collection>> on page 211) |
| IsModified | Returns the modified state of the application. | Get | *Boolean* |
| MatchingPlatformConnectionState | The state if a platform which matches the hardware topology is currently connected. | Get | MatchingPlatformConnectionState (refer to MatchingPlatformConnectionState <<Enumeration>> on page 174) |
| Relations | Returns an object which provides access to the Relations for the ConfigurationDesk automation. | Get | ICaRelations (refer to ICaRelations <<Collection>> on page 160) |
| TransactionCreator | Gets the transaction creator. | Get | ICaTransactionCreator (refer to ICaTransactionCreator <<Interface>> on page 167) |
| WorkingViews | Returns the working views that provide user-defined signal chain segments. | Get | ICaWorkingViews (refer to ICaWorkingViews <<Collection>> on page 171) |

### Methods

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| AddToSelection | Adds objects to the current selection.<br>If parameter Array includes no addable objects no warning will be generated. | ▪ <*System.Array*> `Objects`: The objects to add to the selection. Must not be null.<br>▪ <*Boolean*> `EmptyBefore`: Indicates if current selection should be cleared before adding. Default is false.<br>▪ <*String*> `SelectionType`: The selection type if different selections exists. Default is default selection. | None |
| CanRedo | Determines whether an operation on this instance can be redone. | ▪ <*Signed 32 Bit Integer*> `Count`: The count of operations which should be redone. Default is 1. | *true*if this instance can undo the specified count; otherwise,*false*.<br>▪ *Boolean* |
| CanUndo | Determines whether an operation on this instance can be undone. | ▪ <*Signed 32 Bit Integer*> `Count`: The count of operations which should be undone. Default is 1. | *true*if this instance can undo the specified count; otherwise,*false*.<br>▪ *Boolean* |
| ClearBuildResults | Irreversibly removes existing build results from a | None | None |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| | ConfigurationDesk application and saves the project and application. Throws exception if clear operation fails. | | |
| ConnectObjects | Connects two data objects and creates a mapping between them. If you map ports of model port blocks or device blocks that are not used in the signal chain, the blocks are automatically added to the signal chain. | ▪ <ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)> `First`: First data object of the connection.<br>▪ <ICaDataObject (refer to ICaDataObject <<Interface>> on page 150)> `Second`: Second data object of the connection. | Connection link<br>▪ ICaLink (refer to ICaLink <<Interface>> on page 156) |
| DeleteLink | Deletes the specified link. | ▪ <ICaLink (refer to ICaLink <<Interface>> on page 156)> `Link`: The link which is to delete. | None |
| DeleteLinks | Deletes all links of the specified link array. The specified array must have the dimension of 1. | ▪ *<System.Array>* `Links`: The links. | None |
| Export | Exports the application to an archive. Currently not implemented. | ▪ *<String>* `ArchiveFullPath`: The full path name of the archive where to export the application. | None |
| GetRedoDescriptions | Gets the redo description.If count exceeds number of possible redo operations only the available strings will be returned. | ▪ *<Signed 32 Bit Integer>* `Count`: The count of operation for which the description should be returned. | Collection of strings with description, possible empty.<br>▪ ICaStrings (refer to ICaStrings <<Collection>> on page 165) |
| GetSelectedObjects | Gets the currently selected objects. Currently, only the default selection is supported. Currently, no filter is supported. | ▪ *<String>* `Filter`: Filter to query special objects from the selection. Null or String empty for no filter.<br>▪ *<String>* `SelectionType`: The selection type if different selections exists. Null or String empty for default. | Collection of objects. Maybe empty.<br>▪ ICaObjects (refer to ICaObjects <<Collection>> on page 158) |
| GetUndoDescriptions | Gets the undo descriptions. If count exceeds number of possible undo operations only the available strings will be returned. | ▪ *<Signed 32 Bit Integer>* `Count`: The count of operation for which the description should be returned. | Collection of strings with description, possible empty.<br>▪ ICaStrings (refer to ICaStrings <<Collection>> on page 165) |
| Redo | Redoes an operation on this instance. If redo cannot performed exception will be generated. | ▪ *<Signed 32 Bit Integer>* `Count`: The count of operations which should be redone, default is 1. | None |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| RemoveFromSelection | Removes objects form the current selection. Currently, only the default selection is supported.<br>If parameter Array includes objects which are not removable no warning will be generated. | ▪ *<System.Array>* `Objects`: The objects to remove from the selection. Must not be null.<br>▪ *<String>* `SelectionType`: The selection type if different selections exists. Null or String.Empty for default. | None |
| Rename | Renames the active application. Invalidates this object. Currently not implemented. | ▪ *<String>* `NewApplicationName`: Display name of the new application. | The renamed application, which is now active.<br>▪ *ICaActiveApplication* |
| SaveAs | Saves a copy of the active application and activates the copy. Invalidates this object. Currently not implemented. | ▪ *<String>* `NewApplicationName`: Display name of the new application. | The copy of the application, which is now active.<br>▪ *ICaActiveApplication* |
| Undo | Undoes an operation on this instance. If undo cannot performed exception will be generated. | ▪ *<Signed 32 Bit Integer>* `Count`: The count of operations which should be undone, default is 1. | None |

[1)] <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaActiveProject (refer to ICaActiveProject <<Interface>> on page 206)
- ICaApplication (refer to ICaApplication <<Interface>> on page 208)
- ICaApplicationMain (refer to ICaApplicationMain <<Interface>> on page 192)
- ICaApplications (refer to ICaApplications <<Collection>> on page 209)

# ICaActiveProject <<Interface>>

**Description**

Provides access to the currently active project. If the active project is closed, the corresponding automation object is no longer valid. Access to it can result in an exception.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| ActiveApplication | Returns the active application.<br>May be void. | Get | ICaActiveApplication (refer to ICaActiveApplication <<Interface>> on page 203) |

| Name | Description | Get/Set | Type |
|---|---|---|---|
| Applications | Returns the collection of applications managed by the project. | Get | ICaApplications (refer to ICaApplications <<Collection>> on page 209) |
| Description | Returns or sets the description of the project. | Get/Set | *String* |
| DirectoryName | Returns the folder where the project's CDP file is located. | Get | *String* |
| FileName | Returns the name of the project's CDP file. | Get | *String* |
| Files | Returns the collection of files managed by the project. Currently, the files functionality is not implemented. Count of files collection is -1. | Get | ICaFiles (refer to ICaFiles <<Collection>> on page 211) |
| FullPath | Returns the full path of the project's CDP file. | Get | *String* |
| IsModified | Returns the modified state of the project. | Get | *Boolean* |
| Name | Returns the display name of the project. | Get | *String* |

### Methods

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Backup | Exports the project to an archive. | ▪ <*String*> `ArchivePath`: Full path name of the archive file. | None |
| Close | Closes the project. Invalidates this object. | ▪ <*Boolean*> `SaveChanges`: Determines if to save changes. Default is true. | None |
| RefreshFiles | Synchronizes the files collection with the current file system content.<br>If one moves or deletes files in the project directories, the files collection of the project does not reflect this changes automatically. One must call this method in order to synchronize the collection. | None | None |
| Save | Saves the project. | None | None |
| SaveAs | Saves the project under a different name. Invalidates this object. | ▪ <*String*> `ProjectName`: Name of the new project. | The copy of the project, which is now active.<br>▪ *ICaActiveProject* |
| SaveTo | Lets you save the active project to a target directory.<br>Invalidates this object. | ▪ <*String*> `ProjectName`: Name of the new project.<br>▪ <*String*> `ProjectTargetDirectory` : The target directory of the new project. | The copy of the project, which is now active.<br>▪ *ICaActiveProject* |

1) <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaApplicationMain (refer to ICaApplicationMain <<Interface>> on page 192)
- ICaProject (refer to ICaProject <<Interface>> on page 212)
- ICaProjects (refer to ICaProjects <<Collection>> on page 215)

# ICaApplication <<Interface>>

**Description**

Provides access to an application that is not necessarily active. After removing the application, the object implementing this interface is no longer valid and any access can result in an exception.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| Active | Determines if the application is active. | Get | *Boolean* |
| DirectoryName | Returns the folder where the application's CDL file is located. | Get | *String* |
| FileName | Returns the name of the application's CDL file. | Get | *String* |
| FullPath | Returns the full path of the application's CDL file. | Get | *String* |
| Name | Returns the display name of the application. | Get | *String* |

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Activate | Activates the application. | ▪ *<Boolean>* `AutoSaveActiveApplication`: True to automatically save the current active Application. | The now active Application.<br>▪ ICaActiveApplication (refer to ICaActiveApplication <<Interface>> on page 203) |
| Remove | Removes the application from its project. An application is only removable if it is not active. | ▪ *<Boolean>* `DeleteFromDisk`: Determines if to delete the Applications files from disk. | None |

1) <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaActiveApplication (refer to ICaActiveApplication <<Interface>> on page 203)
- ICaApplications (refer to ICaApplications <<Collection>> on page 209)

## ICaApplications <<Collection>>

**Description**

Provides access to the applications of a project. After removing the project that the applications belong to, the collection is no longer valid and any access results in an exception.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| Count | Returns the number of applications. | Get | *Signed 32 Bit Integer* |

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Add | Create a new application. Returns the new active application. | ▪ *<String>* `ApplicationName`: Display name of new Application.<br>▪ *<Boolean>* `AutoSaveActiveApplication`: Lets you specify whether an active application should be saved. | The Application created, which is now active.<br>▪ ICaActiveApplication (refer to ICaActiveApplication <<Interface>> on page 203) |
| Contains | Tests if an application with the specified name is contained in the collection. | ▪ *<String>* `Name`: Display name of Application searched for. | True if the collection contains an Application with given display name.<br>▪ *Boolean* |
| Import | Imports an application. The imported application becomes the active application. Currently not implemented. Calling this function results in an exception. | ▪ *<String>* `ArchiveFullPath`: Full path name of the Applications archive or Excel file to import.<br>▪ *<String>* `NewApplicationName`: Display name of the imported Application.<br>▪ *<Boolean>* `AutoSaveActiveApplication`: Lets you specify | The imported Application, which is now active.<br>▪ ICaActiveApplication (refer to ICaActiveApplication <<Interface>> on page 203) |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| | | whether an active application should be saved. | |
| Item | Returns an application according to the specified index or display name. | ▪ *<String>* `Index`: Name of the Application or a numeric index value. | The found Application object.<br>▪ ICaApplication (refer to ICaApplication <<Interface>> on page 208) |

[1)] <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaActiveProject (refer to ICaActiveProject <<Interface>> on page 206)
- ICaProject (refer to ICaProject <<Interface>> on page 212)

# ICaFile <<Interface>>

**Description**

Provides access to a file of a project or application.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| DirectoryName | Returns the folder name where the file is located. | Get | *String* |
| Extension | Returns the file name extension of the file. | Get | *String* |
| FileName | Returns the file name. | Get | *String* |
| FullPath | Returns the full path of the file. | Get | *String* |
| Type | Returns the type of the file. | Get | FileType (refer to FileType <<Enumeration>> on page 202) |

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Open | Opens a file. Returns the associated document, for example, layout document or Python document. | None | Document of the file opened.<br>▪ *Object* |
| Remove | Removes the file from its project or application. | ▪ *<Boolean>* `DeleteFromDisk`: | None |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| | Optionally deletes the file from disk. | Determines if the file is deleted from disk. | |

1) <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaFiles (refer to ICaFiles <<Collection>> on page 211)

## ICaFiles <<Collection>>

**Description**

Provides access to the files of a project or application. This interface is intended for future use and is currently not supported. Currently, no files are retrieved from the active application, and the collection is always empty.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| Count | Returns the number of files. This property is currently not supported and returns 0. | Get | *Signed 32 Bit Integer* |

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Contains | Tests if the specified file is contained in the collection. Files are currently not supported. Currently returns an empty collection. | ▪ <*String*> `FileName`: File name of file searched for. | True if the collection contains a file with given file name.<br>▪ *Boolean* |
| Item | Returns a file according to the specified index or file name. This method is currently not supported. No automation object is returned. | ▪ <*String*> `Index`: Name of the file or a numeric index value. | The found file object.<br>▪ ICaFile (refer to ICaFile <<Interface>> on page 210) |

1) <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaActiveApplication (refer to ICaActiveApplication <<Interface>> on page 203)
- ICaActiveProject (refer to ICaActiveProject <<Interface>> on page 206)

# ICaProject <<Interface>>

**Description**

Provides access to a project that is not necessarily active. After removing the project, the object is no longer valid, and the behavior is undefined. Any access then can result in an exception.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| Applications | Returns the collection of the applications of the project. | Get | ICaApplications (refer to ICaApplications <<Collection>> on page 209) |
| FullPath | Returns the full path of the project's CDP file. | Get | *String* |
| Name | Returns the display name of project. | Get | *String* |

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Open | Opens a project. Returns the now active project. | ▪ *<Boolean>* `AutoSaveActiveProject`: True to automatically save the current active project. | The opened project, which is now active.<br>▪ ICaActiveProject (refer to ICaActiveProject <<Interface>> on page 206) |
| Remove | Removes the project from the project's collection. Optionally deletes the whole project from disk. | ▪ *<Boolean>* `PurgeProjectDirectory`: Lets you specify whether the whole project directory shall be purged. | None |

1) <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaProjects (refer to ICaProjects <<Collection>> on page 215)

# ICaProjectManagement <<Interface>>

**Description**

Provides access to the project management.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| AutomaticSaveProjectEnabled | Gets or sets a value indicating whether to enable the automatic saving of a project. | Get/Set | *Boolean* |
| AutomaticSaveProjectInterval | Gets or sets the interval time (in minutes) for automatically saving a project. | Get/Set | *Signed 32 Bit Integer* |
| LoadRecentApplicationOnStartupEnabled | Gets or sets a value indicating whether to load the most recently used application when ConfigurationDesk is started. | Get/Set | *Boolean* |

**Methods**

The element has no methods.

**Event Interfaces**

The element provides the following event interfaces:

- ICaProjectEvents (refer to ICaProjectEvents <<EventInterface>> on page 219)

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaApplicationMain (refer to ICaApplicationMain <<Interface>> on page 192)

# ICaProjectRoot <<Interface>>

**Description**

Provides access to a project root folder and the projects located below it. Each ConfigurationDesk project is related to a project root folder, below which the projects and applications are stored. If a project root folder is removed, the corresponding automation object is no longer valid, and access to it can result in an exception.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| PathName | Gets the full path of the project root folder. | Get | *String* |
| Projects | Returns the projects contained in the project root folder. | Get | ICaProjects (refer to ICaProjects |

| Name | Description | Get/Set | Type |
|---|---|---|---|
| | | | <<Collection>> on page 215) |

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Activate | Makes the project root folder active.<br>The active project root is the one with index 0. This method re-orders the project roots in their collection, they get new indices. | None | None |
| Remove | Removes the project root folder from its collection. | None | None |

1) <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaApplicationMain (refer to ICaApplicationMain <<Interface>> on page 192)
- ICaProjectRoots (refer to ICaProjectRoots <<Collection>> on page 214)
- ICaProjects (refer to ICaProjects <<Collection>> on page 215)

# ICaProjectRoots <<Collection>>

**Description**

Provides access to the project root folders of the application. The collection of project root folders is a collection of ICaProjectRoot objects. Each ConfigurationDesk project is related to a project root folder below which the projects and applications are stored.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| Count | Returns the number of project root folders. | Get | *Signed 32 Bit Integer* |

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Add | Adds a new project root folder. A folder of the full path is created if it does not exist. If the given parameter FullPath contains not a complete and routed path the behavior is undefined. | ▪ *<String>* `FullPath`: Full path name of the new project root directory. | The new project root. ▪ ICaProjectRoot (refer to ICaProjectRoot <<Interface>> on page 213) |
| Contains | Tests if a project root folder with the specified path is contained in the collection. | ▪ *<String>* `PathName`: Full path name of project root searched for. | True if the collection contains a project root with given full path name. ▪ *Boolean* |
| Item | Returns a project root folder according to the specified index or full path. | ▪ *<String>* `Index`: Full path name of the project root or a numeric index value. | The found project root object. ▪ ICaProjectRoot (refer to ICaProjectRoot <<Interface>> on page 213) |

[1)] <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaApplicationMain (refer to ICaApplicationMain <<Interface>> on page 192)

# ICaProjects <<Collection>>

**Description**

Provides access to the projects contained in a project root folder. After removing the project root folder, the projects collection is no longer valid, and the behavior is undefined. Access can then result in an exception.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| Count | Returns the number of projects. | Get | *Signed 32 Bit Integer* |
| ProjectRoot | Returns the root for the projects. | Get | ICaProjectRoot (refer to ICaProjectRoot <<Interface>> on page 213) |

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Add | Creates a new project in the project root folder of this object. The object returned is the now active project.<br>An open project will be closed. | ▪ *<String>* `ProjectName`: Display name of the new project. | The project created, which is now active.<br>▪ ICaActiveProject (refer to ICaActiveProject <<Interface>> on page 206) |
| Contains | Tests if a project with the specified name is contained in the collection. | ▪ *<String>* `Name`: Display name of project searched for. | True if the collection contains a project with given display name.<br>▪ *Boolean* |
| Item | Returns a project according to the specified index or display name. | ▪ *<String>* `Project`: Name of the project or a numeric index value. | The found project object.<br>▪ ICaProject (refer to ICaProject <<Interface>> on page 212) |
| OpenFromBackup | Opens an archived project. If the new project's name is empty, the default name from the archive is used. | ▪ *<String>* `ArchivePath`: Full path name of the projects archive.<br>▪ *<String>* `NewProjectName`: Display name of the new project. If empty the default name from the archive is used.<br>▪ *<Boolean>* `AutoSaveActiveProject`: True to automatically save the current active project. | The project opened, which is now active.<br>▪ ICaActiveProject (refer to ICaActiveProject <<Interface>> on page 206) |

[1)] <Type> `Name`: Description

**Returned by**

The element is returned by properties or methods of the following elements:

- ICaApplicationMain (refer to ICaApplicationMain <<Interface>> on page 192)
- ICaProjectRoot (refer to ICaProjectRoot <<Interface>> on page 213)

# Events

**Where to go from here**

Information in this section

## ICaAutomationEventArgs <<Collection>>

**Description** Provides access to the event arguments of an event of the application.

**Properties** The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| Count | Returns the number of properties in the event arguments collection. | Get | *Signed 32 Bit Integer* |

**Methods** The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| Contains | Tests if a property with the specified name is contained in the collection. | ▪ <*String*> `Name`: The name. | True if the collection contains a property with given name.<br>▪ *Boolean* |
| Item | Returns an event arguments object according to the specified name. | ▪ <*String*> `Name`: The name. | None |
| SetValue | Sets one or more values according to the event that occurred.<br>If the event does not support SetValue or the Name parameter of SetValue isn't known the value will be ignored. | ▪ <*String*> `Name`: The name of the value to set.<br>▪ <*System.Array*> `Values`: The values. | None |

1) <Type> `Name`: Description

## ICaApplicationEvents <<EventInterface>>

**Description** The application's events.

**Methods** The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| BuildFinished | A build process was finished. The build process was either successfully completed or aborted. | ▪ <ICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)> `EArgs`: | None |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| | | TheICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)instance containing the event data. | |
| BuildStarted | A build process was started. The start of the build process cannot be prevented at this stage. | ▪ <ICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)> `EArgs`: TheICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)instance containing the event data. | None |
| BuildStarting | A build process was requested. The start of the build process can be prevented by setting the Cancel event arguments. | ▪ <ICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)> `EArgs`: TheICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)instance containing the event data. | None |
| Quitting | The application is exiting. Any documents, applications and projects are closed. The shutdown of the application can not be prevented at this stage. | None | None |
| Started | The application has started. All components have been loaded and the command line arguments may be accessed. All automation objects may be accessed. If a project has been loaded via command line it can be accessed. This event is useful only for working with the Internal Interpreter. Starting ConfigurationDesk with an external client via dispatch gets the automation object right after ConfigurationDesk's startup process has been finished, so | None | None |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| | that no started event should be necessary. | | |

[1)] <Type> `Name`: Description

**Provided by**

The element is provided by following event sources:

- ICaApplicationMain (refer to ICaApplicationMain <<Interface>> on page 192)

## ICaProjectEvents <<EventInterface>>

**Description**

The project's events.

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| ApplicationClosed | Fired if application was closed. | ▪ <ICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)> `EArgs`: TheICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)instance containing the event data. | None |
| ApplicationClosing | Fired if application will be closed. | ▪ <ICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)> `EArgs`: TheICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)instance containing the event data. | None |
| ApplicationLoaded | Fired if application was loaded. | ▪ <ICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)> `EArgs`: TheICaAutomationEventArgs (refer to ICaAutomationEventArgs | None |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| | | <<Collection>> on page 217)instance containing the event data. | |
| ApplicationLoading | Fired if application will be loaded. | ▪ <ICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)> `EArgs`: TheICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)instance containing the event data. | None |
| ApplicationSaved | Fired after application was saved. | ▪ <ICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)> `EArgs`: TheICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)instance containing the event data. | None |
| ApplicationSaving | Fired before application will be saved. | ▪ <ICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)> `EArgs`: TheICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)instance containing the event data. | None |
| ProjectClosed | Fired if project was closed. | ▪ <ICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)> `EArgs`: TheICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)instance containing the event data. | None |
| ProjectClosing | Fired if project will be closed. | ▪ <ICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)> `EArgs`: | None |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| | | TheICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)instance containing the event data. | |
| ProjectLoaded | Fired if project was loaded. | ▪ <ICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)> **EArgs**: TheICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)instance containing the event data. | None |
| ProjectLoading | Fired if project will be loaded. | ▪ <ICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)> **EArgs**: TheICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)instance containing the event data. | None |
| ProjectRootUpdated | Fired if project root was changed. | ▪ <ICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)> **EArgs**: The parameter EArgs. | None |
| ProjectRootUpdating | Fired if project root is changing. | ▪ <ICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)> **EArgs**: The parameter EArgs. | None |
| ProjectSaved | Fired after Project was saved. | ▪ <ICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)> **EArgs**: TheICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)instance containing the event data. | None |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| ProjectSaving | Fired before Project will be saved. | ▪ <ICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)> `EArgs`: TheICaAutomationEventArgs (refer to ICaAutomationEventArgs <<Collection>> on page 217)instance containing the event data. | None |

[1)] <Type> `Name`: Description

**Provided by**

The element is provided by following event sources:

- ICaProjectManagement (refer to ICaProjectManagement <<Interface>> on page 213)

# ConfigurationDesk Glossary

**Introduction**

The glossary briefly explains the most important expressions and naming conventions used in the ConfigurationDesk documentation.

**Where to go from here**

Information in this section

# A

**Application** There are two types of applications in ConfigurationDesk:

- A part of a ConfigurationDesk project: ConfigurationDesk application.
- An application that can be executed on dSPACE real-time hardware: real-time application.

**Application process** A component of a processing unit application. An application process contains one or more tasks.

**Application process component** A component of an application process. The following application process components are available in the **Components** subfolder of an application process:

- Behavior models that are assigned to the application process, including their predefined tasks, runnable functions, and events.
- Function blocks that are assigned to the application process.

**AutomationDesk** A dSPACE software product for creating and managing any kind of automation tasks. Within the dSPACE tool chain, it is mainly used for automating tests on dSPACE hardware.

**AUTOSAR system description file** An AUTOSAR XML (ARXML) file that describes a system according to AUTOSAR. A system is a combination of a hardware topology, a software architecture, a network communication, and information on the mappings between these elements. The described network communication usually consists of more than one bus system (e.g., CAN, LIN, FlexRay).

# B

**Basic PDU** A general term used in the documentation to address all the PDUs the Bus Manager supports, except for container IPDUs, multiplexed IPDUs, and secured IPDUs. Basic PDUs are represented by the  or  symbol in

tables and browsers. The Bus Manager provides the same functionalities for all basic PDUs, such as ISignal IPDUs or NMPDUs.

**Behavior model** A model that contains the control algorithm for a controller (function prototyping system) or the algorithm of the controlled system (hardware-in-the-loop system). It does not contain I/O functionality nor access to the hardware. Behavior models can be modeled, for example, in MATLAB/Simulink by using Simulink Blocksets and Toolboxes from the MathWorks®.

You can add Simulink behavior models to a ConfigurationDesk application. You can also add code container files containing a behavior model such as Functional Mock-up Units, or Simulink implementation containers to a ConfigurationDesk application.

**Bidirectional signal port** A signal port that is independent of a data direction or current flow. This port is used, for example, to implement bus communication.

**BSC file** A bus simulation container file that is generated with the Bus Manager and contains the configured bus communication of one application process.

**Build Configuration table** A pane that lets you create build configuration sets and configure build settings, for example, build options, or the build and download behavior.

**Build Log Viewer** A pane that displays messages and warnings during the build process.

**Build process** A process that generates an executable real-time application based on your ConfigurationDesk application that can be run on a SCALEXIO system or MicroAutoBox III system. The build process can be controlled and configured via the Build Log Viewer. If the build process is successfully finished, the build result files (build results) are added to the ConfigurationDesk application.

**Build results** The files that are created during the build process. Build results are named after the ConfigurationDesk application and the application process from which they originate. You can access the build results in the Project Manager.

**Bus access** The representation of a run-time communication cluster. By assigning one or more bus access requests to a bus access, you specify which communication clusters form one run-time communication cluster.

In ConfigurationDesk, you can use a bus function block (**CAN**, **LIN**) to implement a bus access. The hardware resource assignment of the bus function block specifies the bus channel that is used for the bus communication.

**Bus access request** The representation of a request regarding the bus access. There are two sources for bus access requests:

- At least one element of a communication cluster is assigned to the **Simulated ECUs**, **Inspection**, or **Manipulation** part of a bus configuration. The related bus access requests contain the requirements for the bus channels that are to be used for the cluster's bus communication.

- A frame gateway is added to the **Gateways** part of a bus configuration. Each frame gateway provides two bus access requests that are required to specify the bus channels for exchanging bus communication.

Bus access requests are automatically included in BSC files. To build a real-time application, each bus access request must be assigned to a bus access.

**Bus Access Requests table** A pane that lets you access bus access requests of a ConfigurationDesk application and assign them to bus accesses.

**Bus configuration** A Bus Manager element that implements bus communication in a ConfigurationDesk application and lets you configure it for simulation, inspection, and/or manipulation purposes. The required bus communication elements must be specified in a communication matrix and assigned to the bus configuration. Additionally, a bus configuration lets you specify gateways for exchanging bus communication between communication clusters. A bus configuration can be accessed via specific tables and its related Bus Configuration function block.

**Bus Configuration Function Ports table** A pane that lets you access and configure function ports of bus configurations.

**Bus Configurations table** A pane that lets you access and configure bus configurations of a ConfigurationDesk application.

**Bus Inspection Features table** A pane that lets you access and configure bus configuration features of a ConfigurationDesk application for inspection purposes.

**Bus Manager**

- Bus Manager in ConfigurationDesk

  A ConfigurationDesk component that lets you configure bus communication and implement it in real-time applications or generate bus simulation containers.
- Bus Manager (stand-alone)

  A dSPACE software product based on ConfigurationDesk that lets you configure bus communication and generate bus simulation containers.

**Bus Manipulation Features table** A pane that lets you access and configure bus configuration features of a ConfigurationDesk application for manipulation purposes.

**Bus simulation container** A container that contains bus communication configured with the Bus Manager. Bus simulation container (BSC) files can be used in the VEOS Player and in ConfigurationDesk. In the VEOS Player, they let you implement the bus communication in an offline simulation application.

In ConfigurationDesk, they let you implement the bus communication in a real-time application independently from the Bus Manager.

**Bus Simulation Features table** A pane that lets you access and configure bus configuration features of a ConfigurationDesk application for simulation purposes.

**Buses Browser** A pane that lets you display and manage the communication matrices of a ConfigurationDesk application. For example, you can access communication matrix elements and assign them to bus configurations. This pane is available only if you work with the Bus Manager.

# C

**Cable harness** A bundle of cables that provides the connection between the I/O connectors of the real-time hardware and the external devices, such as the ECUs to be tested. In ConfigurationDesk, it is represented by an external cable harness component.

**CAFX file** A ConfigurationDesk application fragment file that contains signal chain elements that were exported from a user-defined working view or the Temporary working view of a ConfigurationDesk application. This includes the elements' configuration and the mapping lines between them.

**CDL file** A ConfigurationDesk application file that contains links to all the documents related to an application.

**Channel multiplication** A feature that allows you to enhance the max. current or max. voltage of a single hardware channel by combining several channels. ConfigurationDesk uses a user-defined value to calculate the number of hardware channels needed. Depending on the function block type, channel multiplication is provided either for current enhancement (two or more channels are connected in parallel) or for voltage enhancement (two or more channels are connected in series).

**Channel request** A channel assignment required by a function block. ConfigurationDesk determines the type(s) and number of channels required for a function block according to the assigned channel set, the function block features, the block configuration and the required physical ranges. ConfigurationDesk provides a set of suitable and available hardware resources for each channel request. This set is produced according to the hardware topology added to the active ConfigurationDesk application. You have to assign each channel request to a specific channel of the hardware topology.

**Channel set** A number of channels of the same channel type located on the same I/O board or I/O unit. Channels in a channel set can be combined, for example, to provide a signal with channel multiplication.

**Channel type** A term to indicate all the hardware resources (channels) in the hardware system that provide exactly the same characteristics. Examples for

channel type names: **Flexible In 1**, **Digital Out 3**, **Analog In 1**. An I/O board in a hardware system can have channel sets of several channel types. Channel sets of one channel type can be available on different I/O boards.

**Cluster** Communication cluster.

**Common Program Data folder** A standard folder for application-specific configuration data that is used by all users.

`%PROGRAMDATA%\dSPACE\<InstallationGUID>\<ProductName>`

or

`%PROGRAMDATA%\dSPACE\<ProductName>\<VersionNumber>`

**Communication cluster** A communication network of network nodes that are connected to the same physical channels and share the same bus protocol and address range.

**Communication matrix** A file that defines the communication of a bus network. It can describe the bus communication of one communication cluster or a bus network consisting of different bus systems and clusters. Files of various file formats can be used as a communication matrix: For example, AUTOSAR system description files, DBC files, LDF files, and FIBEX files.

**Communication package** A package that bundles Data Inport blocks which are connected to Data Outport blocks. Hence, it also bundles the signals that are received by these blocks. If Data Inport blocks are executed within the same task and belong to the same communication package, their data inports are read simultaneously. If Data Outport blocks that are connected to the Data Inport blocks are executed in the same task, their output signals are sent simultaneously in one data package. Thus, communication packages guarantee simultaneous signal updates within a running task (data snapshot).

**Configuration port** A port that lets you create the signal chain for the bus communication implemented in a Simulink behavior model. The following configuration ports are available:

- The configuration port of a Configuration Port block.
- The **Configuration** port of a **CAN**, **LIN**, or **FlexRay** function block.

To create the signal chain for bus communication, the configuration port of a **Configuration Port** block must be mapped to the **Configuration** port of a **CAN**, **LIN**, or **FlexRay** function block.

**Configuration Port block** A model port block that is created in ConfigurationDesk during model analysis for each of the following blocks found in the Simulink behavior model:

- **RTICANMM ControllerSetup** block
- **RTILINMM ControllerSetup** block
- **FLEXRAYCONFIG UPDATE** block

Configuration Port blocks are also created for bus simulation containers. A Configuration Port block provides a configuration port that must be mapped

to the Configuration port of a **CAN**, **LIN**, or **FlexRay** function block to create the signal chain for bus communication.

**ConfigurationDesk application** A part of a ConfigurationDesk project that represents a specific implementation. You can work with only one application at a time, and that application must be activated.

An application can contain:

- Device topology
- Hardware topology
- Model topology
- Communication matrices
- External cable harness
- Build results (after a successful build process has finished)

You can also add folders with application-specific files to an application.

**ConfigurationDesk model interface** The part of the model interface that is available in ConfigurationDesk. This specific term is used to explicitly distinguish between the model interface in ConfigurationDesk and the model interface in Simulink.

**Conflict** A result of conflicting configuration settings that is displayed in the Conflicts Viewer. ConfigurationDesk allows flexible configuration without strict constraints. This lets you work more freely, but it can lead to conflicting configuration settings. ConfigurationDesk automatically detects conflicts and provides the **Conflicts Viewer** to display and help resolve them. Before you build a real-time application, you have to resolve at least the most severe conflicts (e.g., errors that abort the build process) to get proper build results.

**Conflicts Viewer** A pane that displays the configuration conflicts that exist in the active ConfigurationDesk application. You can resolve most of the conflicts directly in the **Conflicts Viewer**.

**Container IPDU** A term according to AUTOSAR. An IPDU that contains one or more other IPDUs (i.e., contained IPDUs). When a container IPDU is mapped to a frame, all its contained IPDUs are included in that frame as well.

**ControlDesk** A dSPACE software product for managing, instrumenting and executing experiments for ECU development. ControlDesk also supports calibration, measurement and diagnostics access to ECUs via standardized protocols such as CCP, XCP, and ODX.

**CTLGZ file** A ZIP file that contains a V-ECU implementation. CTLGZ files are exported by TargetLink or SystemDesk. You can add a V-ECU implementation based on a CTLGZ file to the model topology just like adding a Simulink model based on an SLX file.

**Cycle time restriction** A value of a runnable function that indicates the sample time the runnable function requires to achieve correct results. The cycle time restriction is indicated by the **Period** property of the runnable function in the Properties Browser.

# D

**Data inport** A port that supplies data from ConfigurationDesk's function outports to the behavior model.

In a multimodel application, data inports also can be used to provide data from a data outport associated to another behavior model (model communication).

**Data outport** A port that supplies data from behavior model signals to ConfigurationDesk's function inports.

In a multimodel application, data outports also can be used to supply data to a data inport associated to another behavior model (model communication).

**DBC file** A Data Base Container file that describes CAN or LIN bus systems. Because the DBC file format was primarily developed to describe CAN networks, it does not support definitions of LIN masters and schedules.

**Device block** A graphical representation of devices from the device topology in the signal chain. It can be mapped to function blocks via device ports.

**Device connector** A structural element that lets you group device pins in a hierarchy in the External Device Connectors table to represent the structure of the real connector of your external device.

**Device pin** A representation of a connector pin of your external device. Device ports are assigned to device pins. ConfigurationDesk can use the device pin assignment together with the hardware resource assignment and the device port mapping to calculate the external cable harness.

**Device port** An element of a device topology that represents the signal of an external device in ConfigurationDesk.

**Device port group** A structural element of a device topology that can contain device ports and other device port groups.

**Device topology** A component of a ConfigurationDesk application that represents external devices in ConfigurationDesk. You can create a device topology from scratch or easily extend an existing device topology. You can also merge device topologies to extend one. To edit or create device topologies independently of ConfigurationDesk, you can export and import DTFX and XLSX files.

**Documents folder** A standard folder for user-specific documents.

`%USERPROFILE%\Documents\dSPACE\<ProductName>\<VersionNumber>`

**DSA file** A dSPACE archive file that contains a ConfigurationDesk application and all the files belonging to it as one unit. It can later be imported to another ConfigurationDesk project.

**dSPACE Help** The dSPACE online help that contains all the relevant user documentation for dSPACE products. Via the **F1** key or the **Help** button in the dSPACE software you get context-sensitive help on the currently active context.

**dSPACE Log** A collection of errors, warnings, information, questions, and advice issued by all dSPACE products and connected systems over more than one session.

**DTFX file** A device topology export file that contains information on the interface to the external devices, such as the ECUs to be tested. The information includes details of the available device ports, their characteristics, and the assigned pins.

# E

**ECHX file** An external cable harness file that contains the wiring information for the external cable harness. The external cable harness is the connection between the I/O connectors of the real-time hardware and the devices to be tested, for example, ECUs.

**ECU** Abbreviation of *electronic control unit*.

An embedded computer system that consists of at least one CPU and associated peripherals. An ECU contains communication controllers and communication connectors, and usually communicates with other ECUs of a bus network. An ECU can be member of multiple bus systems and communication clusters.

**ECU application** An application that is executed on an ECU. In ECU interfacing scenarios, parts of the ECU application can be accessed (e.g., by a real-time application) for development and testing purposes.

**ECU function** A function of an ECU application that is executed on the ECU. In ECU interfacing scenarios, an ECU function can be accessed by functions that are part of a real-time application, for example.

**ECU Interface Manager** A dSPACE software product for preparing ECU applications for ECU interfacing. The ECU Interface Manager can generate ECU interface container (EIC) files to be used in ConfigurationDesk.

**ECU interfacing** A generic term for methods and tools to read and/or write individual ECU functions and variables of an ECU application. In ECU interfacing scenarios, you can access ECU functions and variables for development and testing purposes while the ECU application is executed on the ECU. For example, you can perform ECU interfacing with SCALEXIO systems or MicroAutoBox III systems to access individual ECU functions by a real-time application.

**EIC file** An ECU interface container file that is generated with the ECU Interface Manager and describes an ECU application that is configured for ECU interfacing. You can import EIC files to ConfigurationDesk to perform ECU interfacing with SCALEXIO systems or MicroAutoBox III systems.

**Electrical interface unit** A segment of a function block that provides the interface to the external devices and to the real-time hardware (via hardware

resource assignment). Each electrical interface unit of a function block usually needs a channel set to be assigned to it.

**Event** A component of a ConfigurationDesk application that triggers the execution of a task. The following event types are available:

- Timer event
- I/O event
- Software event

**Event port** An element of a function block. The event port can be mapped to a runnable function port for modeling an asynchronous task.

**Executable application** The generic term for real-time applications and offline simulation applications. In ConfigurationDesk, an executable application is always a real-time application since ConfigurationDesk does not support offline simulation applications.

**Executable application component** A component of an executable application. The following components can be part of an executable application:

- Imported behavior models including predefined tasks, runnable functions, and events. You can assign these behavior models to application processes via drag & drop or by selecting the **Assign Model** command from the context menu of the relevant application process.
- Function blocks added to your ConfigurationDesk application including associated I/O events. Function blocks are assigned to application processes via their model port mapping.

**Executable Application table** A pane that lets you model executable applications (i.e., real-time applications) and the tasks used in them.

**EXPSWCFG file** An experiment software configuration file that contains configuration data for automotive fieldbus communication. It is created during the build process and contains the data in XML format.

**External cable harness** A component of a ConfigurationDesk application that contains the wiring information for the external cable harness (also known as cable harness). It contains only the logical connections and no additional information such as cable length, cable diameters, dimensions or the arrangement of connection points, etc. It can be calculated by ConfigurationDesk or imported from a file so that you can use an existing cable harness and do not have to build a new one. The wiring information can be exported to an ECHX file or XLSX file.

**External device** A device that is connected to the dSPACE hardware, such as an ECU or external load. The external device topology is the basis for using external devices in the signal chain of a ConfigurationDesk application.

**External Device Browser** A pane that lets you display and manage the device topology of your active ConfigurationDesk application.

**External Device Configuration table** A pane that lets you access and configure the most important properties of device topology elements via table.

**External Device Connectors table** A pane that lets you specify the representation of the physical connectors of your external device including the device pin assignment.

# F

**FIBEX file** An XML file according the ASAM MCD-2 NET standard (also known as Field Bus Exchange Format) defined by ASAM. The file can describe more than one bus system (e.g., CAN, LIN, FlexRay). It is used for data exchange between different tools that work with message-oriented bus communication.

**Find Results Viewer** A pane that displays the results of searches you performed via the Find command.

**FMU file** A Functional Mock-up Unit file that describes and implements the functionality of a model. It is an archive file with the file name extension FMU. The FMU file contains:

- The functionality defined as a set of C functions provided either in source or in binary form.
- The model description file (`modelDescription.xml`) with the description of the interface data.
- Additional resources needed for simulation.

You can add an FMU file to the model topology just like adding a Simulink model based on an SLX file.

**Frame** A piece of information of a bus communication. It contains an arbitrary number of non-overlapping PDUs and the data length code (DLC). CAN frames and LIN frames can contain only one PDU. To exchange a frame via bus channels, a frame triggering is needed.

**Frame triggering** An instance of a frame that is exchanged via a bus channel. It includes transmission information of the frame (e.g., timings, ID, sender, receiver). The requirements regarding the frame triggerings depend on the bus system (CAN, LIN, FlexRay).

**Function block** A graphical representation in the signal chain that is instantiated from a function block type. A function block provides the I/O functionality and the connection to the real-time hardware. It serves as a container for functions, for electrical interface units and their logical signals. The function block's ports (function ports and/or signal ports), provide the interfaces to the neighboring blocks in the signal chain.

**Function block type** A software plug-in that provides a specific I/O functionality. Every function block type has unique features which are different from other function block types.

To use a function block type in your ConfigurationDesk application, you have to create an instance of it. This instance is called a function block. Instances of function block types can be used multiple times in a ConfigurationDesk

application. The types and their instantiated function blocks are displayed in the function library of the Function Browser.

**Function Browser** A pane that displays the function library in a hierarchical tree structure. Function block types are grouped in function classes. Instantiated function blocks are added below the corresponding function block type.

**Function inport** A function port that inputs the values from the behavior model to the function block to be processed by the function.

**Function library** A collection of function block types that allows access to the I/O functionality in ConfigurationDesk. The I/O functionality is based on function block types. The function library provides a structured tree view on the available function block types. It is displayed in the Function Browser.

**Function outport** A function port that outputs the value of a function to be used in the behavior model.

**Function port** An element of a function block that provides the interface to the behavior model via model port blocks.

**Functional Mock-up Unit** An archive file that describes and implements the functionality of a model based on the Functional Mock-up Interface (FMI) standard.

## G

**Global working view** The default working view that always contains all signal chain elements.

## H

**Hardware resource** A hardware element (normally a channel on an I/O board or I/O unit) which is required to execute a function block. A hardware resource can be localized unambiguously in a hardware system. Every hardware resource has specific characteristics. A function block therefore needs a hardware resource that matches the requirements of its functionality. This means that not every function block can be executed on every hardware resource. There could be limitations on a function block's features and/or the physical ranges.

**Hardware resource assignment** An action that assigns the electrical interface unit of a function block to one or more hardware resources. Function blocks can be assigned to any hardware resource which is suitable for

the functionality and available in the hardware topology of your ConfigurationDesk application.

**Hardware Resource Browser** A pane that lets you display and manage all the hardware components of the hardware topology that is contained in your active ConfigurationDesk application in a hierarchical structure.

**Hardware topology** A component of a ConfigurationDesk application that contains information on a specific hardware system which can be used with ConfigurationDesk. It provides information on the components of the system, such as channel type and slot numbers. It can be scanned automatically from a registered platform, created in ConfigurationDesk's Hardware Resource Browser from scratch, or imported from an HTFX file.

**HTFX file** A file containing the hardware topology after an explicit export. It provides information on the components of the system and also on the channel properties, such as board and channel types and slot numbers.

**I/O event** An asynchronous event triggered by I/O functions. You can use I/O events to trigger tasks in your application process asynchronously. You can assign the events to the tasks via drag & drop, via the **Properties Browser** if you have selected a task, or via the **Assign Event** command from the context menu of the relevant task.

**Interface model** A temporary Simulink model that contains blocks from the Model Interface Blockset. ConfigurationDesk initiates the creation of an interface model in Simulink. You can copy the blocks with their identities from the interface model and paste them into an existing Simulink behavior model.

**Interpreter** A pane that lets you run Python scripts and execute line-based commands.

**Inverse model port block** A model port block that has the same configuration (same name, same port groups, and port names) but the inverse data direction as the original model port block from which it was created.

**IOCNET** Abbreviation of I/O carrier network.

A dSPACE proprietary protocol for internal communication in a SCALEXIO system between the real-time processors and I/O units. The IOCNET lets you connect more than 100 I/O nodes and place the parts of your SCALEXIO system long distances apart.

**IPDU** Abbreviation of interaction layer protocol data unit.

A term according to AUTOSAR. An IPDU contains the communication data that is routed from the interaction layer to a lower communication layer and vice

versa. An IPDU can be implemented, for example, as an ISignal IPDU, multiplexed IPDU, or container IPDU.

**ISignal** A term according to AUTOSAR. A signal of the interaction layer that contains communication data as a coded signal value. To transmit the communication data on a bus, ISignals are instantiated and included in ISignal IPDUs.

**ISignal IPDU** A term according to AUTOSAR. An IPDU whose communication data is arranged in ISignals. ISignal IPDUs allow the exchange of ISignals between different network nodes.

## L

**LDF file** A LIN description file that describes networks of the LIN bus system according to the LIN standard.

**LIN master** A member of a LIN communication cluster that is responsible for the timing of LIN bus communication. A LIN master provides one LIN master task and one LIN slave task. The LIN master task transmits frame headers on the bus, and provides LIN schedule tables and LIN collision resolver tables. The LIN slave task transmits frame responses on the bus. A LIN cluster must contain exactly one LIN master.

**LIN schedule table** A table defined for a LIN master that contains the transmission sequence of frame headers on a LIN bus. For each LIN master, several LIN schedule tables can be defined.

**LIN slave** A member of a LIN communication cluster that provides only a LIN slave task. The LIN slave task transmits frame responses on the bus when they are triggered by a frame header. The frame header is sent by a LIN master. A LIN cluster can contain several LIN slaves.

**Local Program Data folder** A standard folder for application-specific configuration data that is used by the current, non-roaming user.

`%USERPROFILE%\AppData\Local\dSPACE\<InstallationGUID>\<ProductName>`

**Logical signal** An element of a function block that combines all the signal ports which belong together to provide the functionality of the signal. Each logical signal causes one or more channel requests. Channel requests are available after you have assigned a channel set to the logical signal.

**Logical signal chain** A term that describes the logical path of a signal between an external device and the behavior model. The main elements of the logical signal chain are represented by different graphical blocks (device blocks, function blocks and model port blocks). Every block has ports to provide the mapping to neighboring blocks.

In the documentation, usually the short form 'signal chain' is used instead.

# M

**MAP file** A file that maps symbolic names to physical addresses.

**Mapping line** A graphical representation of a connection between two ports in the signal chain. You can draw mapping lines in a working view.

**MCD file** A model communication description file that is used to implement a multimodel application. It lets you add several behavior models that were separated from an overall model to the model topology.

The MCD file contains information on the separated models and information on the connections between them. The file is generated with the Model Separation Setup Block in MATLAB/Simulink. The block resides in the Model Interface Blockset (dsmpblib) from dSPACE.

**MDL file** A Simulink model file that contains the behavior model. You can add an MDL file to your ConfigurationDesk application.

As of MATLAB® R2012a, the file name extension for the Simulink model file has been changed from MDL to SLX by The MathWorks®.

**Message Viewer** A pane that displays a history of all error and warning messages that occur during work with ConfigurationDesk.

**Model analysis** A process that analyzes the model to determine the interface of a behavior model. You can select one of the following commands:

- Analyze Simulink Model (Model Interface Only)

  Analyzes the interface of a behavior model. The model topology of your active ConfigurationDesk application is updated with the properties of the analyzed behavior model.
- Analyze Simulink Model (Including Task Information)

  Analyzes the model interface and the elements of the behavior model that are relevant for the task configuration. The task configuration in ConfigurationDesk is then updated accordingly.

**Model Browser** A pane that lets you display and access the model topology of an active ConfigurationDesk application. The **Model Browser** provides access to all the model port blocks available in the behavior models which are linked to a ConfigurationDesk application. The model elements are displayed in a hierarchy, starting with the model roots. Below the model root, all the subsystems containing model port blocks are displayed as well as the associated model port blocks.

**Model communication** The exchange of signal data between the models within a multimodel application. To set up model communication, you must use a mapping line to connect a data outport (sending model) to a data inport

(receiving model). The best way to set up model communication is using the Model Communication Browser.

**Model Communication Browser** A pane that lets you open and browse working views like the **Signal Chain Browser**, but shows only the Data Outport and Data Inport blocks and the mapping lines between them.

**Model Communication Package table** A pane that lets you create and configure model communication packages which are used for model communication in multimodel applications.

**Model implementation** An implementation of a behavior model. It can consist of source code files, precompiled objects or libraries, variable description files and a description of the model's interface. Specific model implementation types are, for example, model implementation containers, such as Functional Mock-up Units or Simulink implementation containers.

**Model implementation container** A file archive that contains a model implementation. Examples are FMUs, SIC files, and VECU files.

**Model interface** An interface that connects ConfigurationDesk with a behavior model. This interface is part of the signal chain and is implemented via model port blocks. The model port blocks in ConfigurationDesk can provide the interface to:

- Model port blocks (from the Model Interface Package for Simulink) in a Simulink behavior model. In this case, the model interface is also called ConfigurationDesk model interface to distinguish it from the Simulink model interface available in the Simulink behavior model.
- Different types of model implementations based on code container files, e.g., Simulink implementation containers, Functional Mock-up Units, and V-ECU implementations.

**Model Interface Package for Simulink** A dSPACE software product that lets you specify the interface of a behavior model that you can directly use in ConfigurationDesk. You can also create a code container file (SIC file) that contains the model code of a Simulink behavior model. The SIC file can be used in ConfigurationDesk and VEOS Player.

**Model port** An element of a model port block. Model ports provide the interface to the function ports and to other model ports (in multimodel applications).

These are the types of model ports:

- Data inport
- Data outport
- Runnable function port
- Configuration port

**Model port block** A graphical representation of the ConfigurationDesk model interface in the signal chain. Model port blocks contain model ports that can be mapped to function blocks to provide the data flow between the function blocks in ConfigurationDesk and the behavior model. The model ports can also be mapped to the model ports of other model port blocks with

data inports or data outports to set up model communication. Model port blocks are available in different types and can provide different port types:

- Data port blocks with data inports and data outports
- Runnable Function blocks with runnable function ports
- Configuration Port blocks with configuration ports. Configuration Port blocks are created during model analysis for each of the following blocks found in the Simulink behavior model:
  - RTICANMM ControllerSetup block
  - RTILINMM ControllerSetup block
  - FLEXRAYCONFIG UPDATE block

  Configuration Port blocks are also created for bus simulation containers.

**Model Separation Setup Block** A block that is contained in the Model Interface Package for Simulink. It is used to separate individual models from an overall model in MATLAB/Simulink. Additionally, model separation generates a model communication description file (MCD file) that contains information on the separated models and their connections. You can use this MCD file in ConfigurationDesk.

**Model topology** A component of a ConfigurationDesk application that contains information on the subsystems and the associated model port blocks of all the behavior models that have been added to a ConfigurationDesk application.

**Model-Function Mapping Browser** A pane that lets you create and update signal chains for Simulink behavior models. It directly connects them to I/O functionality in ConfigurationDesk.

**MTFX file** A file containing a model topology when explicitly exported. The file contains information on the interface to the behavior model, such as the implemented model port blocks including their subsystems and where they are used in the model.

**Multicore real-time application** A real-time application that is executed on several cores of one PU of the real-time hardware.

**Multimodel application** A real-time application that executes several behavior models in parallel on dSPACE real-time hardware (SCALEXIO or MicroAutoBox III).

**Multiplexed IPDU** A term according to AUTOSAR. An IPDU that consists of one dynamic part, a selector field, and one optional static part. Multiplexing is used to transport varying ISignal IPDUs via the same bytes of a multiplexed IPDU.

- The dynamic part is one ISignal IPDU that is selected for transmission at run time. Several ISignal IPDUs can be specified as dynamic part alternatives. One of these alternatives is selected for transmission.

- The selector field value indicates which ISignal IPDU is transmitted in the dynamic part during run time. For each selector field value, there is one corresponding ISignal IPDU of the dynamic part alternatives. The selector field value is evaluated by the receiver of the multiplexed IPDU.
- The static part is one ISignal IPDU that is always transmitted.

**Multi-PU application** Abbreviation of multi-processing-unit application. A multi-PU application is a real-time application that is partitioned into several processing unit applications. Each processing unit application is executed on a separate PU of the real-time hardware. The processing units are connected via IOCNET and can be accessed from the same host PC.

## N

**Navigation bar** An element of ConfigurationDesk's user interface that lets you switch between view sets.

**Network node** A term that describes the bus communication of an ECU for only one communication cluster.

## O

**Offline simulation** A purely PC-based simulation scenario without a connection to a physical system, i.e., neither simulator hardware nor ECU hardware prototypes are needed. Offline simulations are independent from real time and can run on VEOS.

**Offline simulation application** An application that runs on VEOS to perform offline simulation. An offline simulation application can be built with the VEOS Player and the resulting OSA file can be downloaded to VEOS.

**OSA file** An offline simulation application file that is built with the VEOS Player and can be downloaded to VEOS to perform offline simulation.

## P

**Parent port** A port that you can use to map multiple function ports and model ports. All child ports with the same name are mapped. ConfigurationDesk enforces the mapping rules and allows only mapping lines which agree with them.

**PDU** Abbreviation of protocol data unit.

A term according to AUTOSAR. A PDU transports data (e.g., control information or communication data) via one or multiple network layers according to the AUTOSAR layered architecture. Depending on the involved layers and the function of a PDU, various PDU types can be distinguished, e.g., ISignal IPDUs, multiplexed IPDUs, and NMPDUs.

**Physical signal chain** A term that describes the electrical wiring of external devices (ECU and loads) to the I/O boards of the real-time hardware. The physical signal chain includes the external cable harness, the pinouts of the connectors and the internal cable harness.

**Pins and External Wiring table** A pane that lets you access the external wiring information

**Platform** A dSPACE real-time hardware system that can be registered and displayed in the Platform Manager.

**Platform Manager** A pane that lets you handle registered hardware platforms. You can download, start, and stop real-time applications via the Platform Manager. You can also update the firmware of your SCALEXIO system or MicroAutoBox III system.

**Preconfigured application process** An application process that was created via the Create preconfigured application process command. If you use the command, ConfigurationDesk creates new tasks for each runnable function provided by the model which is not assigned to a predefined task. ConfigurationDesk assigns the corresponding runnable function and (for periodic tasks) timer events to the new tasks. The tasks are preconfigured (e.g., DAQ raster name, period).

**Processing Resource Assignment table** A pane that lets you configure and inspect the processing resources in an executable application. This table is useful especially for multi-processing-unit applications.

**Processing unit application** A component of an executable application. A processing unit application contains one or more application processes.

**Project** A container for ConfigurationDesk applications and all project-specific documents. You must define a project or open an existing one to work with ConfigurationDesk. Projects are stored in a project root folder.

**Project Manager** A pane that provides access to ConfigurationDesk projects and applications and all the files they contain.

**Project root folder** A folder on your file system to which ConfigurationDesk saves all project-relevant data, such as the applications and documents of a project. Several projects can use the same project root folder. ConfigurationDesk uses the Documents folder as the default project root

folder. You can specify further project root folders. Each can be made the default project root folder.

**Properties Browser** A pane that lets you access the properties of selected elements.

**PU** Abbreviation of processing unit.

A hardware assembly that consists of a motherboard or a dSPACE processor board, possibly additional interface hardware for connecting I/O boards, and an enclosure, i.e., a SCALEXIO Real-Time PC.

## R

**Real-time application** An application that can be executed in real time on dSPACE real-time hardware. The real-time application is the result of a build process. It can be downloaded to real-time hardware via an RTA file. There are different types of real-time applications:

- Single-core real-time application.
- Multicore real-time application.
- Multi-PU application.

**Restbus simulation** A simulation method to test real ECUs by connecting them to a simulator that simulates the other ECUs in the communication clusters.

**RTA file** A real-time application file. An RTA file is an executable object file for processor boards. It is created during the build process. After the build process it can be downloaded to the real-time hardware.

**Runnable function** A function that is called by a task to compute results. A model implementation provides a runnable function for its base rate task. This runnable function can be executed in a task that is triggered by a timer event. In addition, a Simulink behavior model provides a runnable function for each Hardware-Triggered Runnable Function block contained in the Simulink behavior model. This runnable function is suitable for being executed in an asynchronous task.

**Runnable Function block** A type of model port block. A Runnable Function block provides a runnable function port that can be mapped to an event port of a function block for modeling an asynchronous task.

**Runnable function port** An element of a Runnable Function block. The runnable function port can be mapped to an event port of a function block for modeling an asynchronous task.

**RX** Communication data that is received by a bus member.

# S

**SCALEXIO system** A dSPACE hardware-in-the-loop (HIL) system consisting of one or more real-time industry PCs (PUs), I/O boards, and I/O units. They communicate with each other via the IOCNET. The system simulates the environment to test an ECU. It provides the sensor signals for the ECU, measures the signals of the ECU, and provides the power (battery voltage) for the ECU and a bus interface for restbus simulation.

**SDF file** A system description file that contains information on the CPU name(s), the corresponding object file(s) to be downloaded and the corresponding variable description file(s). It is created during the build process.

**Secured IPDU** A term according to AUTOSAR. An IPDU that secures the payload of another PDU (i.e., authentic IPDU) for secure onboard communication (SecOC). The secured IPDU contains the authentication information that is used to secure the authentic IPDU's payload. If the secured IPDU is configured as a cryptographic IPDU, the secured IPDU and the authentic IPDU are mapped to different frames. If the secured IPDU is not configured as a cryptographic IPDU, the authentic IPDU is directly included in the secured IPDU.

**SIC file** A Simulink implementation container file that contains the model code of a Simulink behavior model. The SIC file can be used in ConfigurationDesk and in VEOS Player.

**Signal chain** A term used in the documentation as a short form for logical signal chain. Do not confuse it with the physical signal chain.

**Signal Chain Browser** A pane that lets you open and browse working views such as the Global working view or user-defined working views.

**Signal inport** A signal port that represents an electrical connection point of a function block which provides signal measurement (= input) functionality.

**Signal outport** A signal port that represents an electrical connection point of a function block which provides signal generation (= output) functionality.

**Signal port** An element of a function block that provides the interface to external devices (e.g., ECUs) via device blocks. It represents an electrical connection point of a function block.

**Signal reference port** A signal port that represents a connection point for the reference potential of inports, outports and bidirectional ports. For example: With differential signals, this is a reference signal, with single-ended signals, it is the ground signal (GND).

**Simulink implementation container** A container that contains the model code of a Simulink behavior model. A Simulink implementation container is generated from a Simulink behavior model by using the Model Interface Package

for Simulink. The file name extension of a Simulink implementation container is SIC.

**Simulink model interface** The part of the model interface that is available in the connected Simulink behavior model.

**Single-core real-time application** An executable application that is executed on only one core of the real-time hardware.

**Single-PU system** Abbreviation of single-processing-unit system.

A system consisting of exactly one PU and the directly connected I/O units and I/O routers.

**SLX file** A Simulink model file that contains the behavior model. You can add an SLX file to your ConfigurationDesk application.

As of MATLAB® R2012a, the file name extension for the Simulink model file has been changed from MDL to SLX by The MathWorks®.

**Software event** An event that is activated from within a task to trigger another subordinate task. Consider the following example: A multi-tasking Simulink behavior model has a base rate task with a sample time = 1 ms and a periodic task with a sample time = 4 ms. In this case, the periodic task is triggered on every fourth execution of the base rate task via a software event. Software events are available in ConfigurationDesk after model analysis.

**Source Code Editor** A Python editor that lets you open and edit Python scripts that you open from or create in a ConfigurationDesk project in a window in the working area. You cannot run a Python script in a **Source Code Editor** window. To run a Python script you can use the **Run Script** command in the Interpreter or on the **Automation** ribbon or the **Run** context menu command in the Project Manager.

**Structured data port** A hierarchically structured port of a Data Inport block or a Data Outport block. Each structured data port consists of more structured and/or unstructured data ports. The structured data ports can consist of signals with different data types (single, double, int8, int16, int32, int64, uint8, uint16, uint32, uint64, Boolean).

**SystemDesk** A dSPACE software product for development of distributed automotive electrics/electronics systems according to the AUTOSAR approach.

SystemDesk is able to provide a V-ECU implementation container (as a VECU file) to be used in ConfigurationDesk.

# T

**Table** A type of pane that offers access to a specific subset of elements and properties of the active ConfigurationDesk application in rows and columns.

**TargetLink** A dSPACE software product for production code generation. It lets you generate highly efficient C code for microcontrollers in electronic control

units (ECUs). It also helps you implement control systems that have been modeled graphically in a Simulink/Stateflow diagram on a production ECU.

TargetLink is able to provide a V-ECU implementation container (as a VECU file) or a Simulink implementation container (SIC file) to be used in ConfigurationDesk.

**Task** A piece of code whose execution is controlled by a real-time operating system (RTOS). A task is usually triggered by an event, and executes one or more runnable functions. In a ConfigurationDesk application, there are predefined tasks that are provided by executable application components. In addition, you can create user-defined tasks that are triggered, for example, by I/O events. Regardless of the type of task, you can configure the tasks by specifying the priority, the DAQ raster name, etc.

**Task Configuration table** A pane that lets you configure the tasks of an executable application.

**Temporary working view** A working view that can be used for drafting a signal chain segment, like a notepad.

**Timer event** A periodic event with a sample rate and an optional offset.

**Topology** A hierarchy that serves as a repository for creating a signal chain. All the elements of a topology can be used in the signal chain, but not every element needs to be used. You can export each topology from and import it to a ConfigurationDesk application. Therefore a topology can be used in several applications.

A ConfigurationDesk application can contain the following topologies:

- Device topology
- Hardware topology
- Model topology

**TRC file** A variable description file that contains all variables (signals and parameters) which can be accessed via the experiment software. It is created during the build process.

**TX** Communication data that is transmitted by a bus member.

# U

**User function** An external function or program that is added to the Automation – User Functions ribbon group for quick and easy access during work with ConfigurationDesk.

# V

**VECU file** A ZIP file that contains a V-ECU implementation. A VECU file can contain data packages for different platforms. VECU files are exported by TargetLink or SystemDesk. You can add a V-ECU implementation based on a VECU file to the model topology in the same way as adding a Simulink model based on an SLX file.

**VEOS** The dSPACE software product for performing offline simulation. VEOS is a PC-based simulation platform which allows offline simulation independently from real time.

**VEOS Player** A software running on the host PC for building offline simulation applications. Offline simulation applications can be downloaded to VEOS to perform offline simulation. ConfigurationDesk lets you generate a bus simulation container (BSC file) via the Bus Manager. You can then import the BSC file into the VEOS Player.

**View set** A specific arrangement of some of ConfigurationDesk's panes. You can switch between view sets by using the navigation bar. ConfigurationDesk provides preconfigured view sets for specific purposes. You can customize existing view sets and create user-defined view sets.

**VSET file** A file that contains all view sets and their settings from the current ConfigurationDesk installation. A VSET file can be exported and imported via the **View Sets** page of the **Customize** dialog.

# W

**Working area** The central area of ConfigurationDesk's user interface.

**Working view** A view of the signal chain elements (blocks, ports, mappings, etc.) used in the active ConfigurationDesk application. A working view can be opened in the Signal Chain Browser or the Model Communication Browser. ConfigurationDesk provides two default working views: The Global working view and the Temporary working view. In the Working View Manager, you can also create user-defined working views that let you focus on specific signal chain segments according to your requirements.

**Working View Manager** A pane that lets you manage the working views of the active ConfigurationDesk application. You can use the **Working View Manager** for creating, renaming, and deleting working views, and also to open a working view in the Signal Chain Browser or the Model Communication Browser.

# X

**XLSX file** A Microsoft Excel™ file format that is used for the following purposes:

- Creating or configuring a device topology outside of ConfigurationDesk.
- Exporting the wiring information for the external cable harness.
- Exporting the configuration data of the currently active ConfigurationDesk application for documentation purposes.

# Appendix

**Where to go from here**

Information in this section

# Introduction to the Message Reader API

**Where to go from here**

Information in this section



## Reading dSPACE Log Messages via the Message Reader API

**Introduction**

You can read log messages of the dSPACE Log via the Message Reader API.

**dSPACE Log**

The dSPACE Log is a collection of errors, warnings, information, questions, and advice issued by all dSPACE products and connected systems over more than one session.

The dSPACE Log is saved as a collection of binary message log files. These files are created when a dSPACE product is running. A single run of a dSPACE product is called a *log session*.

**Note**

If the maximum file size for the binary message log file is reached, messages at the beginning of the dSPACE Log might get deleted. Contact dSPACE Support to solve this.

**Message Reader API**

You can use the Message Reader API to access all binary message log files of the dSPACE Log. You can combine multiple filters to display only log messages according to your specifications. For example, you can configure the Message Reader API to display only log messages from a specific dSPACE product.

The Message Reader API is available as of dSPACE Release 2020-A. For information on the dSPACE products and components that support the Message Reader API, refer to Supported dSPACE Products and Components on page 252.

**dSPACE.Common.MessageReader.dll** The Message Reader API is implemented by the `dSPACE.Common.MessageReader.dll` file. It is located in the `bin` subfolder of the installation folder of each dSPACE product that supports the Message Reader API.

As an exception, the `dSPACE.Common.MessageReader.dll` for the Bus Manager (stand-alone) is located in the `\bin` subfolder of the ConfigurationDesk installation folder.

## Supported dSPACE Releases

The Message Reader API lets you access log messages written by dSPACE products since dSPACE Release 2016-B.

## Message Reader API change in dSPACE Release 2021-A

There is a migration issue specific to the Message Reader API. The issue occurs if you use the API with Python. The issue was caused by the migration to Python 3.9/pythonnet 2.5.3 with dSPACE Release 2021-A.

There is no migration issue to consider if you use the API with C#.

**Specifying a product filter** As of dSPACE Release 2021-A, the `Products` property of the `MessageReaderSettings` class can no longer be used to set the list of products for which to filter in the log sessions. The Message Reader API provides the `SetProducts` method for this purpose.

The following table shows how to specify a product filter before and after migration:

| Using Message Reader API of ... | |
|---|---|
| **... dSPACE Release 2020-B and Earlier (Python 3.6)** | **... dSPACE Release 2021-A and Later (Python 3.9)** |
| `# Specify products whose messages to read:`<br>`Settings = MessageReaderSettings()`<br>`Settings.Products.Add('ControlDesk')`<br>`Settings.Products.Add('AutomationDesk')` | `# Specify products whose messages to read:`<br>`Settings = MessageReaderSettings()`<br>`Settings.SetProducts(['ControlDesk', 'AutomationDesk'])` |

## Related topics

Basics

Supported dSPACE Products and Components ..... 252

Examples

Example of Reading Messages with C# ..... 254
Example of Reading Messages with Python ..... 252

References

MessageReaderSettings Class ..... 261

# Supported dSPACE Products and Components

**Supported dSPACE products and components**

You can use the Message Reader API to access messages from the following dSPACE products and components:

- ASM KnC
- AutomationDesk
- Bus Manager (stand-alone)
- cmdloader
- ConfigurationDesk
- Container Management
- ControlDesk
- dSPACE AUTOSAR Compare
- dSPACE XIL API .NET Implementation
- Firmware Manager
- ModelDesk
- MotionDesk
- Real-Time Testing
- RTI Bypass Blockset
- SYNECT client
- SystemDesk
- TargetLink Property Manager
- VEOS

**Related topics**

Basics

Reading dSPACE Log Messages via the Message Reader API..... 250

# Example of Reading Messages with Python

**Introduction**

You can read the log messages via Python by using the `clr` module. You can combine multiple filters to display only messages according to your specifications.

**Referencing a message reader assembly**

You have to reference a `dSPACE.Common.MessageReader.dll` assembly. For information on the location of the assembly, refer to dSPACE.Common.MessageReader.dll on page 251.

In the following examples it is assumed that the dSPACE Installation Manager is installed and that the message reader assembly is installed in `C:\Program Files\Common Files\dSPACE\InstallationManager\bin`.

The following code references and imports the message reader assembly.

```
# Insert path of message log file access assembly:
import sys
AssemblyPath = r'C:\Program Files\Common Files\dSPACE\InstallationManager\bin'
if not sys.path.count(AssemblyPath):
    sys.path.insert(1, AssemblyPath)

# Add reference to assembly and import it:
import clr
clr.AddReference('dSPACE.Common.MessageReader')
from dSPACE.Common.MessageHandler.Logging import *
```

### Reading all messages

The following example reads all existing message log files and prints all messages via Python. It is assumed that the message reader assembly is referenced and imported. Refer to Referencing a message reader assembly on page 252.

```
# Create message reader and print text of each message:
Reader = MessageReader(None)
for Message in Reader.ReadMessages():
    print(Message.MessageText)
Reader.Dispose()
```

### Filtering messages by severity, product, and session

The following example reads and prints messages with a severity of `Error`, `SevereError`, or `SystemError`. Also, only messages of the last sessions of ControlDesk and AutomationDesk are read and printed. It is assumed that the message reader assembly is referenced and imported. Refer to Referencing a message reader assembly on page 252.

```
# Define error severities:
SEVERITY_ERROR = 3
SEVERITY_SEVERE_ERROR = 4
SEVERITY_SYSTEM_ERROR = 5

# Configure products and sessions whose messages to read:
Settings = MessageReaderSettings()
Settings.MaximalSessionCount = 1
Settings.SetProducts(['ControlDesk', 'AutomationDesk'])

# Create message reader and print text of each error message:
Reader = MessageReader(Settings)
for Message in Reader.ReadMessages():
    # Print error messages only:
    if Message.Severity == SEVERITY_ERROR or \
       Message.Severity == SEVERITY_SEVERE_ERROR or \
       Message.Severity == SEVERITY_SYSTEM_ERROR:
        print('%s: %s' % (Message.Session.ProductName, Message.MessageText))
Reader.Dispose()
```

**Note**

The ReadMessages method returns an enumerator which must either read all messages or must be disposed when no longer used. It is not possible to use two enumerators interleaved, only one enumerator may read messages at a time. Refer to MessageReader Class on page 260.

### Filtering messages by time

Times are given by .NET DateTime objects. Times are given as UTC times (Coordinated Universal Time). You can obtain the current UTC time by `System.DateTime.UtcNow`.

The following example reads all messages after a certain start time. It is assumed that the message reader assembly is referenced and imported. Refer to Referencing a message reader assembly on page 252.

```
import System
Settings = MessageReaderSettings()
Settings.MessageTimeAfter = System.DateTime.UtcNow # Read messages after now

# Create message reader and print time and text of each message:
Reader = MessageReader(Settings)
for Message in Reader.ReadMessages():
    print('%s: %s' % (Message.UtcTimeStamp, Message.MessageText))
Reader.Dispose()
```

### Related topics

Basics

Reading dSPACE Log Messages via the Message Reader API ..... 250
Supported dSPACE Products and Components ..... 252

References

MessageReaderSettings Class ..... 261

## Example of Reading Messages with C#

### Introduction

You can read the log messages via C#. You can combine multiple filters to display only messages according to your specifications.

### Referencing a message reader assembly

You have to reference a `dSPACE.Common.MessageReader.dll` assembly. For information on the location of the assembly, refer to dSPACE.Common.MessageReader.dll on page 251.

### Reading all messages

The following example reads all existing message log files and prints the messages:

```
using dSPACE.Common.MessageHandler.Logging;
...

// Create message reader and print text of each message:
using (MessageReader reader = new MessageReader(null))
{
    foreach (message in reader.ReadMessages())
    {
        Console.WriteLine(message.MessageText);
    }
}
```

### Filtering messages by severity, product, and session

The following example reads and prints messages with a severity of `Error`, `SevereError`, or `SystemError`. Also, only messages of the last sessions of ControlDesk and AutomationDesk are read and printed.

```
using dSPACE.Common.MessageHandler.Logging;
...

// Read the last log sessions of ControlDesk and AutomationDesk only:
MessageReaderSettings settings = new MessageReaderSettings();
settings.MaximalSessionCount = 1;
settings.Products.Add("ControlDesk");
settings.Products.Add("AutomationDesk");

using (MessageReader reader = new MessageReader(settings))
{
    foreach (ILogMessage message in reader.ReadMessages())
    {
        // Print error messages only:
        if (message.Severity == Severity.Error
            || message.Severity == Severity.SevereError
            || message.Severity == Severity.SystemError)
        {
            Console.WriteLine(message.Session.ProductName + ": " + message.MessageText);
        }
    }
}
```

**Note**

The ReadMessages method returns an enumerator which must either read all messages or must be disposed when no longer used. It is not possible to use two enumerators interleaved, only one enumerator may read messages at a time. Refer to MessageReader Class on page 260.

### Related topics

Basics

Reading dSPACE Log Messages via the Message Reader API ..... 250

References

# dSPACE.Common.MessageHandler.Logging Reference

**Where to go from here**

Information in this section



## ILogMessage Interface

**Namespace**

`dSPACE.Common.MessageHandler.Logging`

**Description**

To access information about a message as written to a log file.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| IsStartMessage | Gets a value indicating whether the message is a session start message. | Get | *Boolean* |
| IsStopMessage | Gets a value indicating whether the message is a session stop message. | Get | *Boolean* |
| MainModuleNumber | Gets the main module number of the message. | Get | *Integer* |
| MessageCode | Gets the error code of the message. | Get | *Integer* |
| MessageText | Gets the text of the message. | Get | *String* |
| ModuleName | Gets the module name of the message. | Get | *String* |
| Session | Gets the log session which issued the message. | Get | ILogSession (refer to ILogSession Interface on page 258) |
| Severity | Gets the severity of the message. | Get | Severity (refer to Severity Enumeration on page 263) |

| Name | Description | Get/Set | Type |
|---|---|---|---|
| SubmoduleNumber | Gets the submodule number of the message. | Get | *Integer* |
| ThreadId | Gets the thread ID of the submitting thread. | Get | *Integer* |
| TimeStamp | Gets the time when the message was submitted. Given as local time in the time zone of the session. | Get | *DateTime* |
| UtcTimeStamp | Gets the time when the message was submitted in UTC time. | Get | *DateTime* |

**Methods**

The element has no methods.

**Related topics**

Basics

Reading dSPACE Log Messages via the Message Reader API..... 250

Examples

Example of Reading Messages with C#..... 254
Example of Reading Messages with Python..... 252

References

ILogSession Interface..... 258
Severity Enumeration..... 263

# ILogSession Interface

**Namespace**

`dSPACE.Common.MessageHandler.Logging`

**Description**

To access information about a message log session.

**Properties**

The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| CloseTime | Gets the time when the session was closed. Returns an undefined time (0, DateTimeKind.Unspecified) if the session is still open or was not closed successfully. Given as local time in the time zone of the session. | Get | *DateTime* |

| Name | Description | Get/Set | Type |
|---|---|---|---|
| IsOpen | Gets a value indicating whether the session is still open.<br>If true, the session is still open and new messages can be written. | Get | *Boolean* |
| IsValid | Gets a value indicating whether the session is valid.<br>A session can become invalid if its log files are corrupted. | Get | *Boolean* |
| MetaData | Gets the products metadata as read from log file session info. | Get | *Dictionary< String, String >* |
| ProcessId | Gets the process ID of the log session. | Get | *Integer* |
| ProductName | Gets the product name of the log session. | Get | *String* |
| SessionId | Gets the ID of the log session.<br>This ID is unique in the context of its session reader. | Get | *Integer* |
| StartTime | Gets the sessions start time.<br>Given as local time in the time zone of the session. | Get | *DateTime* |
| TimezoneName | Gets the standard time zone name of the session. | Get | *String* |
| TimezoneOffset | Gets the time zone offset of the session relative to UTC. | Get | *TimeSpan* |
| UtcCloseTime | Gets the time when the session was closed as UTC time.<br>Returns an undefined time (0, DateTimeKind.Unspecified) if the session is still open or was not closed successfully. | Get | *DateTime* |
| UtcStartTime | Gets the start time of the log session as UTC time. | Get | *DateTime* |

## Methods

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| ToSessionTime | Converts UTC time to time zone used when the session was written. | ▪ <*DateTime*> `utcTime`: Specifies the UTC time to convert. | Time in the time zone of the logging session.<br>▪ DateTime |

1) <Type> `Name`: Description

## Related topics

**Basics**

Reading dSPACE Log Messages via the Message Reader API ..... 250

**Examples**

Example of Reading Messages with C# ..... 254
Example of Reading Messages with Python ..... 252

# MessageReader Class

**Description**

To read serialized messages written by dSPACE products.

**Constructor**

The element has the following constructor:

| Name | Description | Parameter[1] | Returns |
|---|---|---|---|
| MessageReader | Initializes a new instance of the MessageReader class. | ▪ <MessageReaderSettings>[2] `settings`: Settings which allow to specify which sessions and messages are read. Can be null, causing all existing log files to be read. | None |

1) <Type> `Name`: Description
2) Refer to MessageReaderSettings Class on page 261

**Properties**

The element has no properties.

**Methods**

The element has the following methods:

| Name | Description | Parameter[1] | Returns |
|---|---|---|---|
| Dispose | Performs application-specific tasks associated with freeing, releasing, or resetting unmanaged resources. | None | None |

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| ReadMessages | Reads the messages written to the log files of the sessions up to now.<br>The messages are returned in chronological order according to their time stamps.<br>**Note**<br>The ReadMessages method returns an enumerator which must either read all messages or must be disposed when no longer used. It is not possible to use two enumerators interleaved, only one enumerator may read messages at a time. | None | Messages read from log file.<br>▪ IEnumerable< ILogMessage (refer to ILogMessage Interface on page 257) > |

[1)] <Type> `Name`: Description

**Related topics**

Basics

Reading dSPACE Log Messages via the Message Reader API ..... 250

Examples

Example of Reading Messages with C# ..... 254
Example of Reading Messages with Python ..... 252

References

MessageReaderSettings Class ..... 261

# MessageReaderSettings Class

**Description**

To define the settings of a message reader.

Used to filter the log sessions and messages read.

**Constructor** The element has the following constructor:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| MessageReaderSettings | Initializes a new instance of the MessageReaderSettings class. | None | None |

1) <Type> `Name`: Description

**Properties** The element has the following properties:

| Name | Description | Get/Set | Type |
|---|---|---|---|
| DirectoryNames | Gets a list of specific directory names from which to read log files.<br>If the list is empty, all standard directories are searched for log files. | Get | *List< String >* |
| MaximalSessionCount | Gets or sets the maximal number of log sessions read for each product.<br>If the count is a positive number n, only the last n sessions are read. If the count is not positive, an unlimited number of sessions is read. The default value is zero, i.e., unlimited. | Get/Set | *Integer* |
| MessageTimeAfter | Gets or sets the minimal time for which messages are read, given as UTC time.<br>Only messages submitted after the message time are read. The message time may be in the past. The message time must be given as valid UTC time. The default time is undefined, i.e., each message time is allowed. | Get/Set | *DateTime* |
| Products | Gets the list of product names for which to read log sessions.<br>If the list is empty sessions of all products are read. | Get | *List< String >* |
| StartTimeAfter | Gets or sets the minimal start time for which sessions are read, given as UTC time.<br>Only sessions which started after the start time are read. The start time may be in the past. The start time must be given as valid UTC time. The default time is undefined, i.e., each start time is allowed. | Get/Set | *DateTime* |

**Methods**

The element has the following methods:

| Name | Description | Parameter[1)] | Returns |
|---|---|---|---|
| SetDirectoryNames | Sets the list of specific directory names from which to read log files.<br>You do not have to specify a list. If the list is empty, all standard directories are searched for log files. | *<string[]>* `names`: Array of directory names. | None |
| SetProducts | Sets the list of product names for which to read log sessions. | *<string[]>* `products`: Array of product names. | None |

1) <Type> `Name`: Description

**Related topics**

Basics

Reading dSPACE Log Messages via the Message Reader API..... 250

Examples

Example of Reading Messages with C#..... 254
Example of Reading Messages with Python..... 252

# Severity Enumeration

**Description**

To specify the severity of a message.

**Enumeration values**

The enumeration has the following values:

| Value | Name | Description |
|---|---|---|
| 0 | Trace | A trace message.<br>Trace messages are usually not created. It depends on the host application if it is possible to configure the message handler to create trace messages. |
| 1 | Info | An information message. |
| 2 | Warning | A warning message. |
| 3 | Error | An error message. |
| 4 | SevereError | A severe error message. |
| 5 | SystemError | A system error message. |
| 6 | Question | A question message. |
| 7 | Advice | An advice message. |

## Related topics

Basics

Reading dSPACE Log Messages via the Message Reader API..... 250

Examples

Example of Reading Messages with C#..... 254
Example of Reading Messages with Python..... 252