

インストール

4D へようこそ！ このページでは、4D 製品のインストールと起動について必要な情報をまとめています。

最低動作環境

4D 製品の macOS / Windows における最小動作環境については、4D Webサイトの [製品ダウンロード](#) ページを参照してください。

追加の情報は 4D Webサイトの [リソースページ](#) にてご確認いただけます。

ディスクへのインストール

4D 製品のインストーラーは 4D の Web サイトから入手していただけます：

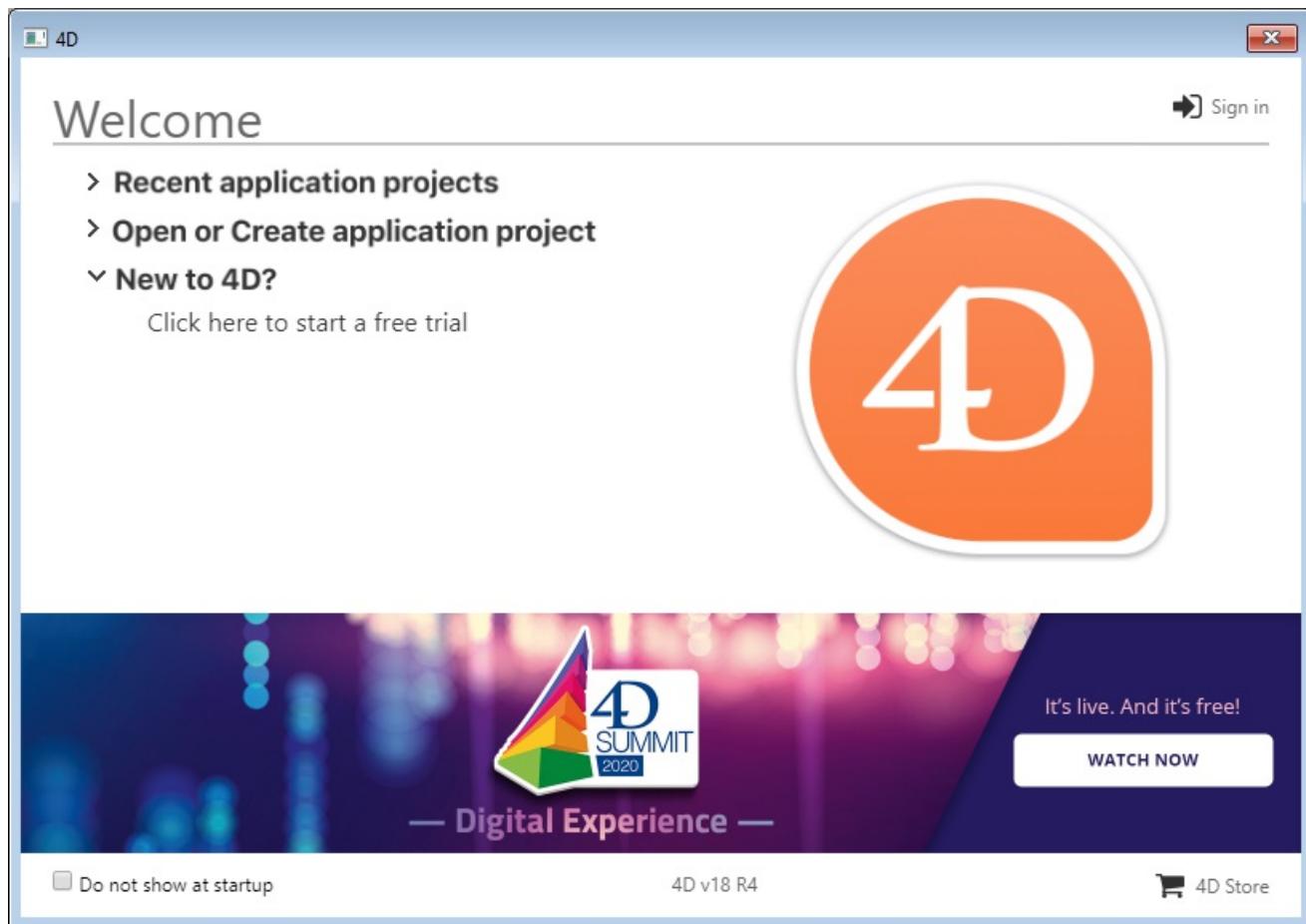
1. 4D Web サイトに接続し、[製品ダウンロード](#) ページを開きます。
2. 必要な製品バージョンのダウンロードリンクをクリックして、インストーラーをダウンロードします。インストールにあたっては、画面に表示される指示に従ってください。

ログイン

インストールが完了すると、4D を起動し、ログインすることができます。起動するには、4D 製品のアイコンをダブルクリックします。



すると、Welcome ウィザードが表示されます：



- 4D を試用するには、"初めて 4D を使いですか？" の下にある、無料体験版を開始するにはここをクリック から先に進みます (4D アカウントでロ

グイン、またはアカウントを新規作成する必要があります)。

- 4D アカウントをすでに持っている場合は、Welcome ウィザードの右上にある ログイン リンクをクリックし、アカウント情報を入力します。マシン上でアクティベーション済みのライセンスは自動的に更新 (追加ライセンスがあれば自動的にロード) されます。

アプリケーションプロジェクトを開始または作成 項目を展開すると、いくつかの選択肢が表示されます:

- 4D Server に接続 - 4D をリモートクライアントとして試用し、4D Server にてすでに起動されているアプリケーションに接続します。
- ローカルアプリケーションプロジェクトを開く - ディスク上に保存されている既存のアプリケーションプロジェクトを開きます。
- 新規にアプリケーションプロジェクトを作成する - ディスク上に新規のアプリケーションプロジェクトを作成します。

ぜひ 4D を楽しんでください !

プロジェクトのアーキテクチャー

4D プロジェクトは、一つの親アプリケーションフォルダー（パッケージフォルダー）に格納された、複数のファイルやフォルダーから構成されています。たとえば：

- MyProject
 - Components
 - Data
 - Logs
 - Settings
 - Documentation
 - Plugins
 - Project
 - DerivedData
 - Sources
 - Trash
 - Resources
 - Settings
 - userPreferences.jSmith
 - WebFolder

バイナリデータベースから変換されたプロジェクトの場合には、追加のフォルダーが存在している場合があります (doc.4d.com にて "データベースをプロジェクトモードに変換する" 参照)。

Project フォルダー

典型的な Project フォルダーの構造です：

- <applicationName>.4DProject ファイル
- Sources
 - Classes
 - DatabaseMethods
 - Methods
 - Forms
 - TableForms
 - Triggers
- DerivedData
- Trash (あれば)

<applicationName>.4DProject ファイル

プロジェクトを定義し、起動するためのプロジェクト開発ファイルです。このファイルを開くには次のいずれかが必要です：

- 4D
- 4D Server (読み取り専用； [リモートプロジェクトを開く](#) 参照)

4D プロジェクトの開発は 4D によっておこない、マルチユーザー開発はソース管理ツールによって管理します。4D Server は .4DProject ファイルを開くことができますが、クライアントからの開発はおこなえません。

このテキストファイルには設定キー（具体的には `"tokenizedText": false`）が含まれる場合があります。

Sources

内容	説明	形式
catalog.4DCatalog	テーブルおよびフィールド定義	XML
folders.json	エクスプローラーフォルダー定義	JSON
menus.json	メニュー定義	JSON
settings.4DSettings	ストラクチャーデータベース設定。 ユーザー設定 または データファイル用のユーザー設定 が定義されている場合は、そちらの設定が優先されます。 警告: コンパイル済みアプリケーションの場合、ストラクチャーデータベースは読み取り専用の .4dz ファイルに格納されます。運用時にカスタム設定を定義するには、ユーザー設定 または データファイル用のユーザー設定 を使う必要があります。	XML
tips.json	定義されたヘルプTips	JSON
lists.json	定義されたリスト	JSON
filters.json	定義されたフィルター	JSON
styleSheets.css	CSS スタイルシート	CSS
styleSheets_mac.css	Mac用 CSS スタイルシート (変換されたバイナリデータベースより)	CSS
styleSheets_windows.css	Windows用 CSS スタイルシート (変換されたバイナリデータベースより)	CSS

DatabaseMethods

内容	説明	形式
databaseMethodName.4dm	プロジェクト内で定義されているデータベースメソッド (1つのデータベースメソッドにつき1ファイル)。	テキスト

Methods

内容	説明	形式
methodName.4dm	プロジェクト内で定義されているプロジェクトメソッド (1つのメソッドにつき1ファイル)。	テキスト

Classes

内容	説明	形式
className.4dm	特定のオブジェクトをインスタンス化するための、ユーザークラス用の定義メソッド。 1クラスにつき1ファイル。ファイル名がクラス名になります。	テキスト

Forms

内容	説明	形式
formName/form.4DForm	プロジェクトフォームの定義	json
formName/method.4dm	プロジェクトフォームメソッド	テキスト
formName/Images/pictureName	プロジェクトフォームのスタイルシート	picture
formName/ObjectMethods/objectName.4dm	オブジェクトメソッド (1つのオブジェクトメソッドにつき1ファイル)	テキスト

TableForms

内容	説明	形式
<i>n/Input/formName/form.4DForm</i>	入力テーブルフォームの定義 (n: テーブル番号)	json
<i>n/Input/formName/Images/pictureName</i>	入力テーブルフォームのスタティックピクチャー	picture
<i>n/Input/formName/method.4dm</i>	入力テーブルフォームのフォームメソッド	テキスト
<i>n/Input/formName/ObjectMethods/ObjectName.4dm</i>	入力テーブルフォームのオブジェクトメソッド (1つのオブジェクトメソッドにつき1ファイル)	テキスト
<i>n/Output/formName/form.4DForm</i>	出力テーブルフォーム (n: テーブル番号)	json
<i>n/Output/formName/Images/pictureName</i>	出力テーブルフォームのスタティックピクチャー	picture
<i>n/Output/formName/method.4dm</i>	出力テーブルフォームのフォームメソッド	テキスト
<i>n/Output/formName/ObjectMethods/ObjectName.4dm</i>	出力テーブルフォームのオブジェクトメソッド (1つのオブジェクトメソッドにつき1ファイル)	テキスト

Triggers

内容	説明	形式
table_n.4dm	プロジェクト内で定義されているトリガーメソッド (1つのテーブルにつき1ファイル ; n: テーブル番号)	テキスト

注: 拡張子 .4dm のファイルは、4D メソッドのコードをテキスト形式で格納しており、ソース管理ツールに対応しています。

Trash

プロジェクトから削除されたメソッドやフォームがあれば、Trash フォルダーにはそれらが格納されます。たとえば、つぎのフォルダーが格納されている場合があります:

- Methods
- Forms
- TableForms

削除された要素はファイル名に括弧が付いた形でフォルダー内に置かれます (例: "(myMethod).4dm")。フォルダーの構成は [Sources](#) フォルダーと同じです。

DerivedData

DerivedData フォルダーには、処理を最適化するため 4D が内部的に使用するキャッシュデーターが格納されます。これらは必要に応じて自動的に生成・再生成されます。このフォルダーは無視してかまいません。

Libraries

このフォルダーは macOS でのみ使用されます。

Libraries フォルダーには、macOS 上で [Apple Silicon用にコンパイル](#) された結果のファイルが格納されます。

Resources

Resources フォルダーには、追加のカスタムプロジェクトリソースファイルやフォルダーが格納されます。アプリケーションインターフェースの翻訳やカスタマイズに必要なファイルはすべてここに格納します (ピクチャー、テキスト、XLIFF ファイルなど)。4D は自動のメカニズムによってフォルダー内のファイル (とくに XLIFF ファイルおよびスタティックピクチャー) を扱います。リモートモードにおいては、サーバーとすべてのクライアントマシン間でファイルを共有することができます (4D Server リファレンスマニュアルの [リソースフォルダの管理](#) を参照ください)。

内容	説明	形式
item	プロジェクトリソースファイルとフォルダー	様々
Images/Library/item	ピクチャーライブラリの個別ピクチャーファイル(*)。各アイテムの名称がファイル名となります。名称が重複する場合には、名称に番号が追加されます。	picture

(*) .4db バイナリデータベースから変換されたプロジェクトの場合のみ

Data

Data フォルダーには、データファイルのほか、データに関わるするファイルやフォルダーがすべて格納されています。

内容	説明	形式
data.4dd(*)	レコードとして入力されたデータと、レコードに属するデータが格納されたデータファイルです。4D プロジェクトを開くと、アプリケーションはカレントデータファイルをデフォルトで開きます。このファイルの場所や名称を変更した場合は、データファイルを開くダイアログボックスが表示され、使用するデータファイルを選択するか、新しいデータファイルを作成できます。	バイナリ
data.journal	データベースがログファイルを使用する場合のみ作成されます。ログファイルは2つのバックアップ間のデータ保護を確実なものにするために使用されます。データに対して実行されたすべての処理が、このファイルに順番に記録されます。つまりデータに対して操作がおこなわれるたびに、データ上の処理（操作の実行）とログファイル上の処理（操作の記録）という2つの処理が同時に発生します。ログファイルはユーザーの処理を妨げたり遅くしたりすることなく、独立して構築されます。データベースは1つのログファイルしか同時に使用できません。ログファイルにはレコードの追加・更新・削除やトランザクションなどの処理が記録されます。ログファイルはデータベースが作成される際にデフォルトで生成されます。	バイナリ
data.match	（内部用）テーブル番号に対応する UUID	XML

(*) .4db バイナリデータベースからプロジェクトに変換した場合、データファイルは変換による影響を受けません。このデータファイルの名称を変更して移動させることができます。

Settings

Settings フォルダーには、アプリケーションの管理に使用される データファイル用のユーザー設定 ファイルが格納されます。

この設定は [ユーザー設定](#) や [ストラクチャー設定](#) より優先されます。

内容	説明	形式
directory.json	このデータファイルを使ってアプリケーションが実行されている場合に使用する 4D グループとユーザー、およびアクセス権の定義	JSON
Backup.4DSettings	このデータファイルを使ってデータベースが実行されている場合に使用する バックアップオプション を定義したデータベースバックアップ設定です。バックアップ設定に使われるキーについての説明は バックアップ設定ファイル マニュアルを参照ください。	XML
settings.4DSettings	データファイル用のカスタムデータベース設定。	XML

Logs

Logs フォルダーには、プロジェクトが使用するすべてのログファイルが格納されます。以下のログファイルが格納されます：

- データベース変換
- Webサーバーリクエスト
- バックアップ/復元アクションのジャーナル (*Backup Journal[xxx].txt*、[バックアップジャーナル](#) 参照)
- コマンドログ
- 4D Serverリクエスト (クライアントマシンおよびサーバー上で生成)

データフォルダーが読み取り専用モードの場合やメンテナンスログファイルの保存には、システムのユーザー設定フォルダー (Active 4D Folder

のこと、詳しくは [Get 4D folder コマンド参照](#)) 内にある Logs フォルダーが利用されます。

Settings

Settings フォルダーには、アプリケーションの管理に使用される ユーザー設定 ファイルが格納されます。

この設定は [ストラクチャー設定](#) より優先されます。ただし、[データファイル用のユーザー設定ファイル](#) が存在する場合には、そちらが優先されます。

内容	説明	形式
directory.json	4D グループとユーザー、およびアクセス権の定義	JSON
Backup.4DSettings	バックアップ開始時に バックアップオプション を指定するためのデータベースバックアップ設定です。このファイルは、バックアップジャーナル に保存する情報量などの追加オプションの確認や設定にも使用することができます。バックアップ設定に使われるキーについての説明は バックアップ設定ファイル マニュアルを参照ください。	XML
BuildApp.4DSettings	アプリケーションビルダーのダイアログボックス、または <code>BUILD APPLICATION</code> コマンドを使ったときに自動的に作成されるビルド設定ファイル	XML
settings.4DSettings	プロジェクト用のカスタム設定 (すべてのデータファイル)	XML

userPreferences.<userName>

ブレークポイントやウィンドウの位置など、ユーザーの環境設定を定義するファイルを格納するフォルダーです。このフォルダーは無視してかまいません。格納されるファイルの例です：

内容	説明	形式
methodPreferences.json	カレントユーザーのメソッドエディター環境設定	JSON
methodWindowPositions.json	カレントユーザーのメソッドのウィンドウポジション	JSON
formWindowPositions.json	カレントユーザーのフォームのウィンドウポジション	JSON
workspace.json	開かれているウィンドウのリスト；macOS ではタブウィンドウの順序	JSON
debuggerCatches.json	キャッチコマンドリスト	JSON
recentTables.json	最近開かれたテーブルのリスト	JSON
preferences.4DPreferences	カレントデータパスおよび主なウィンドウの位置	XML
CompilerIntermediateFiles	Apple Silicon用にコンパイルした結果生成される中間ファイル	Folder

Components

アプリケーションプロジェクトが利用するコンポーネントを格納するフォルダーです。このフォルダーは、Project フォルダーと同じ階層に置きます。

アプリケーションプロジェクトはコンポーネントとして利用することができます：

- 開発においては、ホストプロジェクトの Components フォルダーに .4dproject ファイルのエイリアスを置きます。 - 運用においては、[コンポーネントをビルド](#) し、生成された .4dz ファイルを .4dbase フォルダーに格納し、それをホストアプリケーションの Components フォルダーに置きます。

Plugins

アプリケーションプロジェクトが利用するプラグインを格納するフォルダーです。このフォルダーは、Project フォルダーと同じ階層に置きます。

Documentation

このフォルダーには、クラス・メソッド・フォームなどのプロジェクト要素について作成されたドキュメンテーションファイル (.md) がすべて格納されます。ドキュメンテーションファイルは、4D エクスプローラーにて表示・管理されます。

詳細については [プロジェクトのドキュメンテーション](#) を参照ください。

WebFolder

ページ、ピクチャーなどのための、4D Web サーバーのデフォルトのルートフォルダー。Web サーバーが初回起動時に、自動で作成されます。

.gitignore ファイル (任意)

git が無視するファイルを指定します。プロジェクトに gitignore ファイルを含めるには、環境設定 > 一般 ページの .gitignore ファイルを作成するオプションを使用します。このファイルの内容を設定するには、[.gitignore ファイルを作成する](#) を参照してください。

ドキュメンテーション

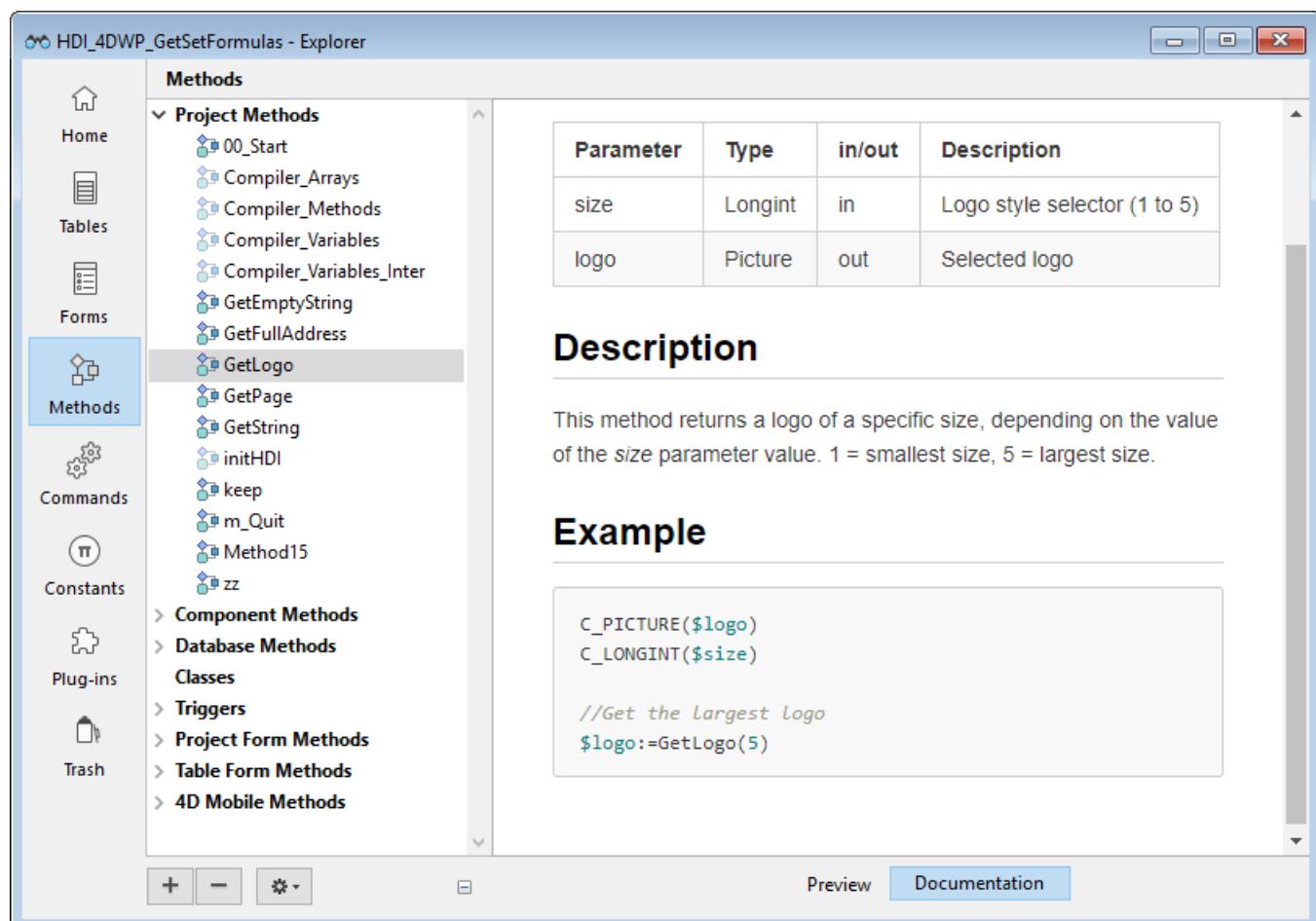
アプリケーションプロジェクトにおいては、メソッドやフォーム、テーブル、フィールドに関するドキュメンテーションを作成することができます。複数のプログラマーによってプロジェクトを開発している場合などに、ドキュメンテーションの作成はとくに適しています。また、一般的に良いプログラミングの作法としても推奨されます。ドキュメンテーションには、要素の説明だけでなく、アプリケーションにおけるその要素の機能を理解するために必要なあらゆる情報を含めることができます。

ドキュメントすることができるプロジェクト要素は次のとおりです:

- メソッド (データベースメソッド、コンポーネントメソッド、プロジェクトメソッド、フォームメソッド、4D Mobile メソッド、トリガー、クラス)
- フォーム
- テーブルとフィールド

ドキュメンテーションファイルは Markdown 記法 (.md ファイル) で記述します。これには、Markdown をサポートしている任意のエディターを使うことができます。これらはそれぞれ独立したファイルとしてプロジェクトフォルダー内に格納されます。

ドキュメントされた内容は、エクスプローラーの右側にあるプレビューエリアに表示されます:



また、[コードエディターのヘルプTip](#) として部分的に表示することもできます。

ドキュメンテーションファイル

ドキュメンテーションファイル名

ドキュメンテーションファイルには、ドキュメントの対象である要素と同じファイル名が付き、拡張子は ".md" です。たとえば、`myMethod.4dm` プロジェクトメソッドに付随するドキュメンテーションファイルの名前は `myMethod.md` です。

エクスプローラー上では、選択した要素と同じファイル名のドキュメンテーションが自動的に表示されます (後述参照)。

ドキュメンテーションファイルのアーキテクチャー

ドキュメンテーションファイルはすべて、データベースフォルダーのルートにある `Documentation` フォルダーに格納されます。

`Documentation` フォルダーのアーキテクチャーは次のとおりです:

- `Documentation`
 - `Classes`
 - `myClass.md`
 - `DatabaseMethods`
 - `onStartup.md`
 - ...
 - `Forms`
 - `loginDial.md`
 - ...
 - `Methods`
 - `myMethod.md`
 - ...
 - `TableForms`
 - 1
 - `input.md`
 - ...
 - ...
 - `Triggers`
 - `table1.md`
 - ...
- プロジェクトフォームとそのプロジェクトフォームメソッドは、同じドキュメンテーションファイルをフォームとメソッドの両方について共有します。
- テーブルフォームとそのテーブルフォームメソッドは、同じドキュメンテーションファイルをフォームとメソッドの両方について共有します。

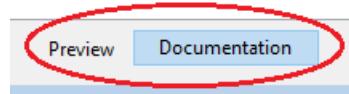
ドキュメントされているプロジェクト要素を名称変更したり、削除したりすると、その要素に紐づいている Markdown ファイルも自動で名称変更、または削除されます。

エクスプローラーとドキュメンテーション

ドキュメンテーションの表示

エクスプローラーウィンドウにドキュメンテーションを表示させるには:

1. プレビュー領域が表示されていることを確認します。
2. エクスプローラーリストより、ドキュメントされている要素を選択します。
3. プレビュー領域の下にある ドキュメンテーション ボタンをクリックします。



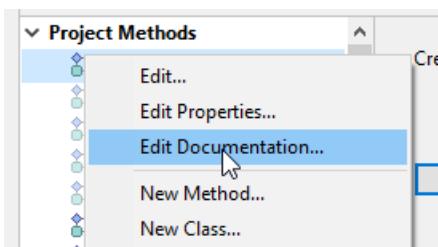
- 選択要素のドキュメンテーションファイルが見つからなかった場合には、作成する ボタンが表示されます。
- 選択要素のドキュメンテーションファイルが存在すれば、その内容がエリア内に表示されます。なお、エリアに表示されている内容は直接編集することはできません。

ドキュメンテーションファイルの編集

選択要素の Markdown ドキュメンテーションファイルはエクスプローラーより作成・編集することができます。

選択要素のドキュメンテーションファイルが存在しなければ:

- Documentation ペインにある 作成する ボタンをクリックするか、
- エクスプローラーのオプションメニューまたはコンテキストメニューより ドキュメンテーションを編集... を選択します。

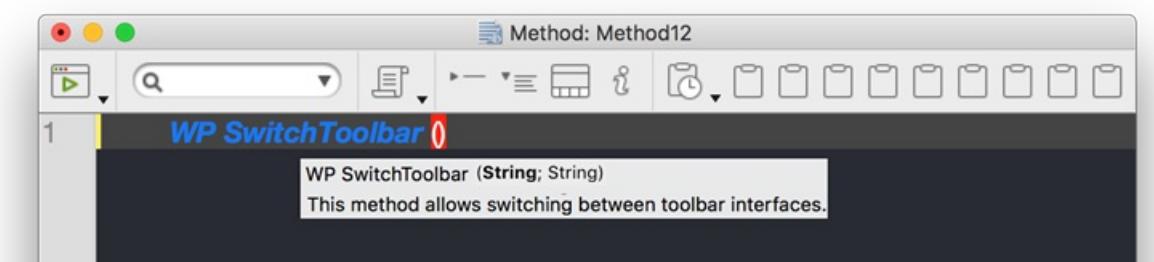


テンプレートを使い、適切な場所・名称で自動生成された .md ファイルは、デフォルトの Markdown エディターで開かれます。

選択要素のドキュメンテーションファイルが存在していれば、エクスプローラーのオプションメニューまたはコンテキストメニューより ドキュメンテーションを編集... を選択することで、Markdown エディターに開くことができます。

コードエディターでドキュメンテーションを表示する

4D のコードエディターは、メソッドのドキュメンテーションの一部をヘルプTip として表示します。



"<MethodName>.md" ファイルが "<package>/documentation" フォルダーに存在する場合、コードエディターは次の優先順位でヘルプTip を表示します:

- Markdown ファイルの先頭に設置した、HTML コメントタグで囲まれたテキスト (<!-- コマンドの説明 -->)
- HTML のコメントタグが使用されていなければ、Markdown ファイルの # Description タグ後の最初の文章
この場合、最初の文章には 4D コードパーサーによって自動生成されたメソッドのプロトタイプが入ります。

それ以外の場合には、[メソッドコードの先頭のコメントブロック](#) がコードエディターに表示されます。

ドキュメンテーションファイルの定義

4D はテンプレートを用いて新規のドキュメンテーションファイルを作成します。このテンプレートは、[コードエディターでドキュメンテーションを表示する](#) のに利用できる項目が提供しています。

それ以外の [サポートされている Markdown タグ](#) も利用することができます。

新規作成されたドキュメンテーションファイルには、次のデフォルト項目が含まれています:

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

線	説明
"<!-- Type your summary here -->"	HTML コメントタグ。メソッドの説明として優先的に コードエディターTip に表示されます。
## Description	Markdown のレベル2 見出しタグ。HTML コメントタグが使用されていない場合、このタグ後の最初の文章がメソッドの説明としてコードエディターTip に表示されます。
## Example	レベル2 見出しタグ。サンプルコードの記述に使用できます。
` 4D Type your example here \`	4D サンプルコードのフォーマットに使います (highlight.js ライブリを使用)。

サポートされている Markdown

- 見出しタグ:

```
# 見出し 1
## 見出し 2
### 見出し 3
```

- スタイルタグ (イタリック、太字、取り消し線) :

```
_イタリック_
**太字**
**_太字/イタリック_**
~~取り消し線~~
```

- 4D コードハイライトが付くコードブロックタグ (```4d ... ```)

```
` 4d C_TEXT($txt) $txt:="Hello world!" \`
```

- テーブルタグ:

引数	型	説明
wpArea	文字列	Write pro エリア
toolbar	文字列	ツールバー名

- リンクタグ:

```
// 例 1
コマンドの [ドキュメンテーション](https://doc.4d.com) は ...

// 例 2
[4D ブログ] [1]

[1]: https://blog.4d.com
```

- 画像タグ:

```
![画像の説明](pictures/image.png)

![4D ロゴ](https://blog.4d.com/wp-content/uploads/2016/09/logo0rignal-1.png "4D blog logo")

![4D ブログのロゴとリンク](https://blog.4d.com/wp-content/uploads/2016/09/logo0rignal-1.png "4D blog logo")
```



詳細については [GitHub Markdown guide](#) (英文) を参照ください。

例題

WP SwitchToolbar.md ファイルに、次のように書くことができます:

```
<!-- size 引数に応じて、異なるロゴを返します -->

GetLogo (size) -> logo

| 引数 | 型 | in/out | 説明 |
| ----- | ----- | ----- | ----- |
| size | 倍長整数 | in | ロゴスタイルセレクター (1 から 5) |
| logo | ピクチャー | out | 選択されたロゴ |

## Description

このメソッドは、*size* 引数の値に応じて、特定サイズのロゴを返します。
1 = 最小値, 5 = 最大値

## Example

C_PICTURE($logo)
C_LONGINT($size)

// 最大ロゴを取得します
$logo:=GetLogo(5)
```

- エクスプローラーの表示:

HDL_4DWP_GetSetFormulas - Explorer

Methods

Project Methods

- 00_Start
- Compiler_Arrays
- Compiler_Methods
- Compiler_Variables
- Compiler_Variables_Inter
- GetEmptyString
- GetFullAddress
- GetLogo**
- GetPage
- GetString
- initHDI
- keep
- m_Quit
- Method15
- zz

Component Methods

Database Methods

Classes

Triggers

Project Form Methods

Table Form Methods

4D Mobile Methods

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

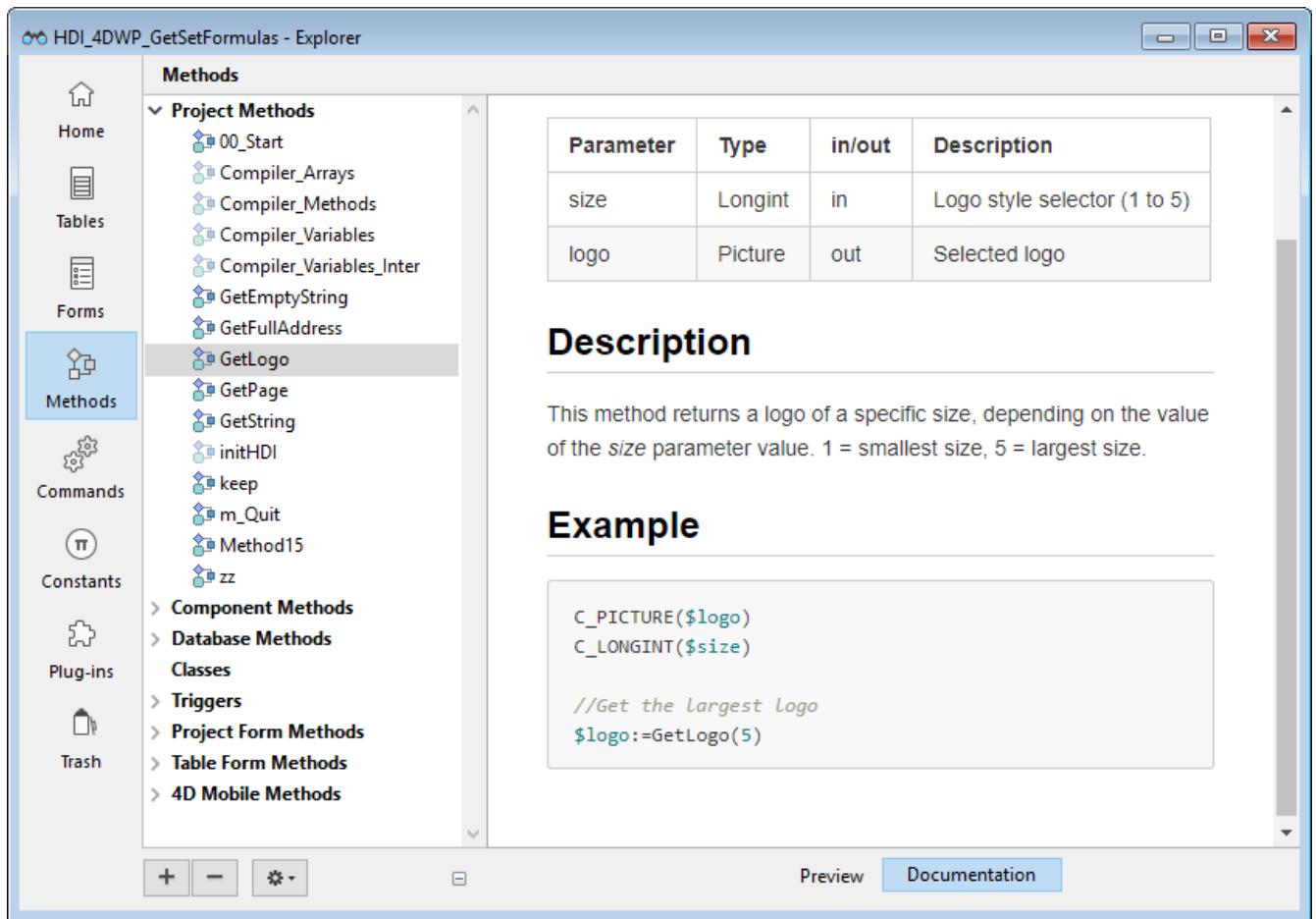
Description

This method returns a logo of a specific size, depending on the value of the *size* parameter value. 1 = smallest size, 5 = largest size.

Example

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

Preview Documentation



- コードエディターの表示:

[GetLogo](#)

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

4D ランゲージについて

4Dは独自のプログラミング言語を持っています。1300を超えるこのビルトインの言語により、4Dは Web、モバイル、およびデスクトップアプリケーションを作成する、パワフルな開発ツールとなっています。単純な計算から複雑なカスタムインターフェースの作成まで、4D 言語は様々なタスクに使用することができます。たとえば、以下のようなことが可能です：

- クエリや並べ替えなどのレコード管理エディターにプログラムでアクセスする
- データベースの情報をもとに複雑なレポートやラベルを作成・印刷する
- 他のデバイスと通信する
- メールを送信する
- ドキュメントや Web サイトを管理する
- 4D アプリケーションと他のアプリケーションの間で、データの書き出しや読み込みをおこなう
- 4D のプログラミング言語に、他の言語で書かれたプロシージャーを組み込む

4D のプログラミング言語は柔軟性とパワーを備え、あらゆるレベルのユーザー・デベロッパーにとって多種多様な情報管理業務を達成するための理想的なツールです。初心者のユーザーであっても、計算処理を手早く実行できます。ある程度コンピューターの知識を持っているユーザーであれば、プログラミング経験がなくてもアプリケーションをカスタマイズできます。熟練したデベロッパーであれば、4D の強力なプログラミング言語を駆使して、ファイル転送や通信、モニタリングなどの高度な機能をアプリケーションに組み込むことができます。他言語でプログラミング経験がある開発者は、独自のコマンドを 4D に追加することができます。

4D ランゲージとは

4D ランゲージは、私たちが日ごろ話している言語とされて変わりありません。アイデアの表現や、伝達、指示をおこなうためのコミュニケーションの一形態です。話し言葉と同様に、4D は独自の語彙・文法・構文を持っています。このランゲージを使用して、アプリケーションやデータをどのように扱うのかを 4D に伝えます。

4D を効果的に使用する目的では、ランゲージのすべてを知っている必要はありません。言葉を交わすために言語のすべてを知る必要がないのと同じです。実際、少ない語彙でも雄弁に語ることはできるものです。4D ランゲージも、創造性を發揮するのに必要となるのはほんの一部で、残りは必要に応じて覚えればよいのです。

なぜランゲージを使用するのか

4D を使いはじめると、最初はプログラミング言語があまり必要ないように思えるかもしれません。プログラミングせずとも広範囲のデータ管理タスクをおこなうことのできる自由度の高いツールを、4D はデザインモードにおいて提供しています。データ入力やクエリ、並べ替え、レポート作成などの基本的なタスクは簡単に処理できます。それだけでなく、データ検証や入力補助、グラフ作成、ラベルの生成など、多くの追加機能も提供されています。

では、なぜ 4D ランゲージが必要なのでしょうか。たとえば、以下のようないくつかの用途が考えられます：

- 自動的な繰り返しタスク：データの更新、複雑なレポートの生成、長い一連の操作などを全自动でおこなうことができます。
- ユーザーインターフェースのコントロール：ウィンドウおよびメニューの管理、フォームやインターフェースオブジェクトの制御ができます。
- 高度なデータ管理：トランザクション処理、複雑なデータ検証、マルチユーザー管理、セットや命名セレクションの処理などが含まれます。
- コンピューターのコントロール：シリアルポート通信やドキュメント管理、エラー管理が可能です。
- アプリケーションの作成：ダブルクリックで起動する、カスタマイズされたアプリケーションをビルドできます。
- ビルトイン 4D Web サーバーの利用：データを反映させた動的な Web ページをビルドし、更新できます。

ランゲージを使用すれば、アプリケーションのデザインや処理を完全に制御することができます。4D はパワフルな汎用エディターを提供していますが、必要に応じてアプリケーションをカスタマイズするには 4D ランゲージが必要です。

データ制御

4D ランゲージを使用すれば、パワフルでエレガントに、データをコントロールできます。4D ランゲージは初心者にも使いやすく、経験豊かなデベロッパーの利用に耐えるほど高度です。ランゲージの利用により、ビルトインのデータベース機能から、完全にカスタマイズされたアプリケーションにスムーズに移行できます。

4D ランゲージのコマンドを使用して標準のレコード管理エディターにアクセスできます。たとえば、デザインモードにおいて「クエリ」ツールバーのボタンを使って開けるクエリエディターは、`QUERY` コマンドを使用することでも表示されます。さらに、`QUERY` コマンドを使用すれば、エディターを開かなくても指定した

データを検索できるのです。たとえば `QUERY ([People]; [People]Last Name="Smith")` と記述すると、データベースから "Smith" という名前の人をすべて検索します。

4D ランゲージはとてもパワフルです。ひとつのコマンドが、従来のプログラム言語で書かれた数百行あるいは数千行にも及ぶコードに相当することもしばしばです。このパワーを持ちながら、コマンドにはシンプルな英語の名称がつけられています。例えはクエリを行うには `QUERY` コマンドを、レコードを追加するには `ADD RECORD` コマンドを使用します。

4D ランゲージは、ほとんどのタスクを簡単に実行できるようデザインされています。レコードの追加、並べ替え、データの検索などの操作がシンプルなコマンドで提供されています。さらにコマンドを使用してシリアルポートのコントロール、ディスク上のドキュメントの読み込み、高度なトランザクション処理等を行うこともできるのです。

4D ランゲージは非常に高度なタスクをも比較的簡単にこなします。このようなタスクをランゲージなしでおこなうことはほとんど想像できません。ランゲージのパワフルなコマンドを使用したとしても、タスクによっては複雑で難しいものとなることがあります。ツール自身はタスクを処理しませんが、困難なタスクの処理を助けてくれます。たとえば、ワープロを使用すれば早く簡単に文章を書くことができますが、ワープロ自体が文章を代わりに考えてくれるわけではありません。4D ランゲージの使用はデータ管理を楽にし、複雑なタスクにも自信を持って取り掛かれるようにします。

4D ランゲージは従来のコンピュータ言語ですか

従来のコンピュータ言語に馴れ親しんでいる方は、本節を参照してください。それ以外の方は、この章を読みとばしても構いません。

4D ランゲージは従来のコンピュータ言語とは異なります。4D ランゲージは、今日のコンピュータで使用できる最も先進的で柔軟性のある言語の1つです。4D ランゲージは、人が作業をおこなうように処理するよう設計されています。

従来の言語を使用して開発を行う場合、まず念入りに計画を立てる必要があります。実際、計画の立案は開発の重要な工程の1つです。4Dにおいては、プロジェクトのあらゆる部分でいつでもランゲージを使いはじめることができます。たとえば、初めにフォームにメソッドを追加し、いくつかのメソッドをあとから追加することができます。アプリケーションがより高度になったら、メニューから実行するプロジェクトメソッドを追加することもできます。4D ランゲージは好きだけ利用することができます。他の多くのデータベースのような "すべてか無か" ではありません。

従来の言語では、インターフェースオブジェクトをあらかじめ正式な記述法で宣言・定義しなければなりません。しかし、4D ではボタンなどのオブジェクトを作成し、そのまま使用することができます。作成したオブジェクトは自動的に 4D によって管理されます。たとえば、ボタンを使用するためには、ボタンをフォーム上に作成して名前を指定します。ユーザーがボタンをクリックした時点で、ランゲージが自動的にメソッドに通知します。

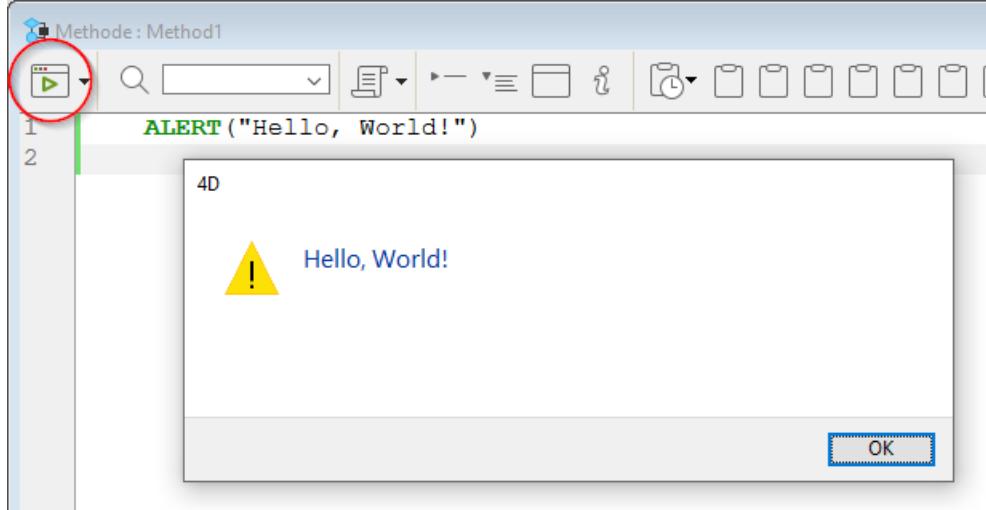
従来の言語は、コマンドの記述方法を制限するなどして、融通性に欠けることが多いのに対し、4D ランゲージは伝統に縛られることなく進化しています。

概要

4D ランゲージを使用して "Hello, world!" メッセージを表示するには複数の方法があります。一番簡単な方法はおそらく、プロジェクトメソッドにコードを1行、次のように書くやり方です：

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

このコードは、"Hello, World!" メッセージが表示された、OK ボタンの付いたプラットフォームの標準的なアラートダイアログボックスを開きます。コードを実行するには、メソッドエディターの左上にある実行ボタンをクリックします：



あるいは、フォーム内のボタンにこのコードを付けた場合、フォームを実行した状態でボタンをクリックすると、その都度アラートメッセージが表示されます。いずれの方法でも、前述の1行のコードを実行するだけで目的達成です！

値の代入

変数、フィールド、配列要素などを対象に、データを格納したり、格納したデータを別の対象にコピーしたりすることができます。変数にデータを格納することを、変数にデータを代入すると言い、代入演算子 (:=) を使っておこないます。代入演算子はフィールドや配列要素に対してデータを代入する場合にも使います。

```
$MyNumber:=3 // MyNumber 変数に数値の3を代入します
[Products]Size:=$MyNumber // [Products]Size フィールドに MyNumber 変数の値を代入します
arrDays{2}:="Tuesday" // arrDays 配列の第二要素に文字列 "Tuesday" を代入します
MyVar:=Length("Acme") // MyVar 変数に関数の結果（数値の4）を代入します
$myDate:=!2018/01/21! // 日付リテラルを代入します
$myHour:=?08:12:55? // 時間リテラルを代入します
```

代入演算子 (:=) は必ず他の演算と区別しなければなりません。代入演算子は、被演算子を組み合わせて新しい一つのものにするのではなく、演算子の右側の式の値を左側の変数やフィールドにコピーします。

重要: 代入演算子 (:=) と比較演算子 (=) を混同しないように注意してください。(=) とは異なる代入演算子が採用されたのは意図的なことで、他のプログラミング言語で (==) や (==>) の使用によって度々起こる間違いを避けるためです。このような間違いはコンパイラにとても発見しにくく、時間を消耗するトラブルシューティングのもとです。

変数

4D ランゲージは強い型付けの言語ですが、多くの場合に柔軟性も発揮します。型指定の変数を作成するには `var` キーワードを使います。たとえば、日付型の変数を作成するには、次のように書くことができます：

```
var MyDate : Date
```

var キーワードを使って、定義されているクラス型のオブジェクト変数を宣言することができます。例：

```
var myPerson : cs.Person  
// Person ユーザークラスの変数
```

推奨はされませんが、変数を使用することで宣言することもでき、必ずしも正式に宣言する必要はありません。たとえば、今日の日付に30日足した値を格納した変数が欲しい場合、次のように書くことができます：

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

上のコードは "MyOtherDate に、現在の日付に30日を加算した値を代入します" という意味です。この1行で変数が宣言され、変数に（仮の）データ型とデータが割り当てられます。このように代入によって宣言された変数はデータ型が規定されていないと解釈され、コードの違う行では別のデータ型の値を代入することもでき、その際にはデータ型を動的に変化させます。 var によって宣言された変数はデータ型を変化させることはできません。コンパイルモードにおいては、その宣言方法にかかわらず、変数のデータ型は変更できません。

コマンド

4D コマンドとは、処理を実行するために 4D に組み込まれている命令文のことです。すべての 4D コマンド、たとえば CREATE RECORD や ALERT などのコマンドはテーマ別に 4D ランゲージリファレンス に記載されています。コマンドに引数を渡す場合は、コマンド名の後の括弧 () に引数を入れ、セミコロン (;) で区切れます。例：

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

コレクションやオブジェクトにコマンドが属している場合、それらは名前付きメソッドであり、ドット記法を用いて使用します。たとえば：

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

4D プラグインや 4D コンポーネントを利用して、4D 開発環境に新しくコマンドを追加することもできます。

4D のユーザーコミュニティーや、サードパーティーデベロッパーによるプラグインが多数存在します。たとえば、[4d-plugin-pdf-pages](#) プラグインを macOS で使用した場合は次のコードが書けます：

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

4D SVG はアプリケーションの機能を拡張するユーティリティコンポーネントの一例です：

```
// 図の描画  
svgRef:=SVG_New  
objectRef:=SVG_New_arc(svgRef;100;100;90;90;180)
```

4D SVG は 4D に含まれています。

定数

4D では多くの定義済定数が用意されており、それらの値は名前によってアクセスすることができます。定義済みの定数によって、より可読性の高いコードを書くことができます。たとえば、Read Mode は定数で、その値は 2 です。

```
vRef:=Open document("PassFile";"TEXT";Read Mode) // ドキュメントを読み取り専用モードで開きます
```

メソッドエディターにおいて、定義済定数はデフォルトで 下線付き で表示されます。

メソッド

4D が提供するたくさんのビルトインコマンドを使って、独自の プロジェクトメソッド を組み立てることができます。プロジェクトメソッドとはユーザー定義のメソッドで、コマンドや演算子などの要素からなり立ちます。プロジェクトメソッドは汎用性のあるメソッドですが、そうではない他の種類のメソッドも存在します：オブジェクトメソッド、フォームメソッド、テーブルメソッド（トリガー）、データベースメソッド。

メソッドは、一つ以上のステートメントで構成されます。ステートメントとは、メソッドの1行のことで1つの命令を実行します。ステートメントは単純な場合もあれば、複雑な場合もあります。

たとえば、次のステートメントは確認ダイアログボックスを表示します：

```
CONFIRM("このアカウントを本当に閉じますか?";"はい";"いいえ")
```

メソッドは、テストとループの制御フローの実行を含みます。 `If...Else...End if` および `Case of...Else...End case` の分岐構造が使用できるほか、ループ構造としては `While...End while`、`Repeat...Until`、`For...End for`、そして `For each...End for each` が使用可能です：

テキスト変数 `vtSomeText` の文字を一つ一つループ処理します：

```
For($vlChar;1;Length(vtSomeText))
    // 文字がタブであれば
    If(Character code(vtSomeText[$vlChar])=Tab)
        // なんらかの処理をします
    End if
End for
```

プロジェクトメソッドは他のプロジェクトメソッドを呼び出すことができ、その際に引数を渡すことも可能です。メソッドに引数を渡す場合は、メソッド名の後の括弧 () に引数を入れ、セミコロン (;) で区切れます。引数は受け取り側のメソッドにて、受け取り順に番号が付けられたローカル変数 (\$1, \$2, ...\$n) に格納されます。メソッドの一つの値を戻り値とすることができます、\$0 パラメーターを使います。メソッドを呼び出すには、メソッド名を書きます：

```
$myText:="hello"
$myText:=Do_Something($myText) // Do_Something メソッドを呼び出します
ALERT($myText) //"HELLO"

// Do_Something メソッドのコードです
$0:=Uppercase($1)
```

データタイプ

4D ランゲージで扱うデータにはいくつかの種別があり、これらのデータ種別を "データタイプ" と呼びます。基本のデータタイプ（文字、数値、日付、時間、ブール、ピクチャー、ポインター、配列）と混合型のデータタイプ（BLOB、オブジェクト、コレクション）があります。

データタイプのうち、文字タイプと数値タイプは、複数の類似するフィールドタイプに対応する点に注意してください。これらのフィールドにデータが格納されるとき、4D ランゲージはフィールドタイプに合致するデータタイプへとデータを自動的に変換します。反対に、たとえば整数フィールドのデータを呼び出すと、そのデータは自動的に数値タイプとして扱われます。つまり、4D ランゲージを使用する際に、類似するフィールドタイプを厳密に区別する必要はありません。

しかし、プログラミングにおいて異なるデータタイプを混同しないようにすることは重要です。"ABC" を日付フィールドに格納しても意味がないように、日付型の変数に "ABC" を格納することも意味がありません。4D は、コードに書かれたことをできるだけ有効にしようとします。たとえば、日付に数値を加算した場合は、日付に日数を加算したいものと認識します。しかし、日付に文字列を加算した場合には、4D はその操作が意味を持たないことを警告します。

あるタイプとして格納したデータを、別のタイプとして使用する場合があります。4D ランゲージには、データタイプを変換するためのコマンドが用意されています。たとえば、数値で始まり、"abc" 等の文字で終了する部品番号を作成する場合、以下のように記述することができます:

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

数値変数 Number の値が17であれば、[Products]Part_Number に "17abc" という文字列が代入されます。

データタイプについては [データタイプ](#) の節で詳しく説明しています。

オブジェクトとコレクション

4D ランゲージのオブジェクトとコレクションは、オブジェクト記法を使用して値を代入・取得することができます。たとえば:

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

大カッコ内と文字列の組み合わせを用いることもできます:

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

オブジェクトプロパティ値には、オブジェクトやコレクションも設定することができます。これらのサブプロパティにアクセスするため、オブジェクト記法では連続した字句を受け入れることができます:

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

オブジェクトのプロパティ値が、メソッド（フォーミュラ）をカプセル化したオブジェクトである場合には、プロパティ名の後に括弧（）をつけることで実行できます:

```
$f:=New object  
$f.message:=New formula(ALERT("Hello world!"))  
$f.message() // "Hello world!" を表示します
```

コレクションの要素にアクセスするためには、大カッコでくくった要素番号を渡します:

```
C_COLLECTION(myColl)  
myColl:=New collection("A";"B";1;2;Current time)  
myColl[3] // コレクションの4番目の要素にアクセスします（0起点）
```

クラス

4D ランゲージではオブジェクトクラスがサポートされています。"myClass" という名称のクラスを作成するには、プロジェクトの Project/Sources/Classes フォルダーに `myClass.4dm` ファイルを追加します。

あるメソッドにおいて、クラスのオブジェクトをインスタンス化するには、クラストア（`cs`）よりユーザークラスを呼び出して、`new()` メンバー関数を使います。引数を渡すこともできます。

```
// 4D メソッド内  
$o:=cs.myClass.new()
```

`myClass` クラスマソッド内では、`methodName` クラスメンバーメソッドを宣言するのに `Function <methodName>` ステートメントを使います。ほかのメソッドのように、クラスメンバーメソッドは引数を受け取ったり、値を返すことができ、オブジェクトインスタンスとして `This` を使えます。

```
// myClass.4dm ファイル内
Function hello
C_TEXT($0)
$0:="Hello "+This.who
```

クラスメンバーメソッドを実行するには、オブジェクトインスタンスのメンバーメソッドに `()` 演算子を使います。

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

`Class constructor` キーワードを使用してオブジェクトのプロパティを宣言することもできます（任意）。

```
// Rectangle.4dm ファイル内
Class constructor
C_LONGINT($1;$2)
This.height:=$1
This.width:=$2
This.name:="Rectangle"
```

クラスはほかのクラスから継承することもできます：`Class extends <ClassName>`。また、`Super` コマンドを使って、スーパークラスを呼び出すことができます。たとえば：

```
// Square.4dm ファイル内
Class extends Rectangle

Class constructor
C_LONGINT($1)

// 親クラスのコンストラクターを呼び出します
// 長方形の高さ・幅パラメーターに正方形の一辺の長さを引数として渡します
Super($1;$1)

This.name:="Square"
```

演算子

プログラミング言語を使用する際に、データのみを必要とする場合は非常に稀です。データを加工、または何らかの目的のために使用することがほとんどです。そういった計算は演算子を使っておこないます。一般的に演算子とは、2つのデータをもとに処理をおこない、1つの新しいデータを生成します。日常的に使用されている演算子も多くあります。例えば、`1 + 2` という式は加算演算子（プラス記号）を使用し、2つの数値を足し合わせて、`3`という結果を返します。以下に、よく知られている4つの演算子を示します。

演算子	演算子	例題
+	加算（足し算）	<code>1 + 2</code> の結果は <code>3</code>
-	減算（引き算）	<code>3 - 2</code> の結果は <code>1</code>
*	乗算（かけ算）	<code>2 * 3</code> の結果は <code>6</code>
/	除算（割り算）	<code>6 / 2</code> の結果は <code>3</code>

数値演算子は、使用可能な演算子のうちの1種にすぎません。4Dは、数値・テキスト・日付・ピクチャ等、異なるタイプのデータを扱うために、各データタイプで演算を実行するための演算子を備えています。

対象のデータタイプによって、同じ記号が異なる処理に使用される場合があります。例えば、データタイプによってプラス記号（+）は下記のように異なる演算を実行します：

データタイプ	演算子	例題
数値	加算 (足し算)	<code>1 + 2</code> は数値を加算し、結果は 3 です。
String	連結 (結合)	<code>"みなさん" + "こんにちは"</code> は文字を連結 (結合) し、結果は <code>"みなさんこんにちは"</code> です。
日付と数値	日付の加算	<code>!2006/12/4! + 20</code> は、2006年12月4日に 20日を加算し、結果は 2006年12月24日です。

式

式は、値を返します。4D ランゲージでコードを書く際には、意識していなくても常に式を使用しています。式は、"フォーミュラ" と呼ぶこともあります。

コマンド・演算子・変数・フィールド・オブジェクトプロパティ・コレクション要素等、複数のランゲージの要素を組み合わせて式は構成されます。式により、ステートメント (メソッドの 1文や 1行) を構成します。データが必要なとき、式が必要になります。

式が単独で使われることはほとんどありませんが、単独で使用できる場合がいくつかあります：

- フォーミュラエディター (フォーミュラによるクリエイティブ替えなど)
- `EXECUTE FORMULA` コマンド
- フォームオブジェクトやウィジェットのデータソースとして
- デバッガー内で式の値を確認することができます
- クイックレポートエディターでカラムにフォーミュラを使用することができます

式のタイプ

生成する値のタイプによって、式のタイプを定義することができます。式のタイプは複数あります。様々なタイプの式の例を以下に示します。

式	タイプ	説明
"こんにちは"	文字列	これは文字列定数 "こんにちは" です。文字列定数であることを表すために二重引用符が必要です。
"みなさん" + "こんにちは"	文字列	2つの文字列 "みなさん" と "こんにちは" が + 演算子により結合され、"みなさんこんにちは" を返します。
[People]Name + "様"	文字列	2つの文字列の結合です。[People]Name フィールドと文字列 "様" が結合されます。フィールドの値が "小林" の場合、"小林様" を返します。
Uppercase ("smith")	文字列	この式は Uppercase コマンドを使用して、文字列 "smith" を英大文字に変換します。そして "SMITH" を返します。
4	数値	これは数値定数 4です。
4 * 2	数値	2つの数値、4 と 2 の乗算です。乗算演算子の (*) を使用しています。数値の 8を返します。
myButton	数値	これはボタンに紐づけられた変数です。ボタンの現在の値を返します：クリックされた場合に 1、それ以外は 0 を返します。
!06/12/24! または !2006/12/24!	日付	この式は日付定数で 2006年12月24日を表します。
Current date + 30	日付	これは日付の計算です。Current date コマンドは現在の日付を返します。現在の日付に 30 日を加えた日付を返します。
?8:05:30?	時間	これは時間定数で、8時5分30秒を表します。
?2:03:04? + ?1:02:03?	時間	2つの時間の足し算をおこない、3時5分7秒を返します。
True	ブール	このコマンドはブール値の true (真) を返します。
10 # 20	ブール	これは 2つの数値の論理比較です。#記号は、"等しくない" を表します。10と20は "等しくない" ため、この式は true (真) を返します。
"ABC" = "XYZ"	ブール	これは文字列の論理比較です。文字列は等しくないため、式は FALSE (偽) を返します。
My Picture + 50	ピクチャー	この式は My Picture 変数に入っているピクチャーを右に 50ピクセル移動したピクチャーを返します。
->[People]Name	ポインター	この式は [People]Name フィールドへのポインターを返します。
Table (1)	ポインター	このコマンドは一番目に定義されたテーブルへのポインターを返します。
JSON Parse (MyString)	オブジェクト	このコマンドは MyString が適切なフォーマットであれば、オブジェクトとして返します。
JSON Parse (MyJSONArray)	コレクション	このコマンドは MyJSONArray が適切なフォーマットであれば、コレクションとして返します。
Form.pageNumber	オブジェクトプロパティ	オブジェクトプロパティは式として、サポートされているいずれのタイプでもあります。
Col[5]	コレクション要素	コレクション要素は式として、サポートされているいずれのタイプでもあります。
\$entitySel[0]	エンティティ	ORDA のエンティティセレクションの要素である、エンティティを返します。これは 代入不可の式 です。

代入可 vs 代入不可の式

式は、数値の4や"Hello" の文字列のようなリテラル定数であったり、\$myButton のような変数であったりします。式には演算子も含まれられます。たとえば、4 + 2 という式は加算演算子を使って二つの数値を加算し、結果の 6 を返します。リテラル定数や演算子を使った式は 代入不可の式で、式に値を代入することはできません。代入可能な式 も存在します。代入演算子の左側に使えるものが、代入可能な式です。たとえば：

```
// 変数 $myVar は代入可能です:  
$myVar:="Hello" // $myVar に "Hello" を代入します  
//Form.pageNumber は代入可能です:  
Form.pageNumber:=10 // Form.pageNumber に 10 を代入します  
//Form.pageTotal-Form.pageNumber は代入不可です:  
Form.pageTotal- Form.pageNumber:=10 // 代入不可のため、エラー
```

このように、リテラル定数ではなくても、演算子を使っている式は代入不可です。たとえば、`[Person]FirstName+" "+[Person]LastName` は代入不可です。

ポインター

ポインターは、プログラミングにおいてデータを参照するための高度な方法です。4D ではテーブル、フィールド、変数、配列、配列要素を参照するためにポインターを使用することができます。

対象へのポインターは、その対象の前にポインター記号 (`->`) を付けることで取得することができます。反対にポインターから対象を取得するには、ポインター名の後にポインター記号をつけます:

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

コメント

コメントとは、コード内の実行されないテキストのことです。これらのテキストは、コード実行時にインターペリターによって無視されます。

コメントの書き方は2通りあります:

- `//` 記号の後はすべてコメントとして扱われるため、これを使って1行のコメントが書けます
- `/*コメント*/` の表記方法でインラインコメント、または複数行にまたがるコメントが書けます

これらの書き方は同時に使用できます。

シングルラインコメント (//)

コードの後や行の最初に `//` を使うと、その後のテキストはすべてコメントとなります。例:

```
// これはコメントです  
For($vCounter;1;100) // ループを開始します  
  // コメント  
  // コメント  
  // コメント  
End for
```

インライン、およびマルチラインコメント /* */

コメントを `/*` と `*/` で囲むと、そのあいだのテキストはコメントとなります。この方法でインラインおよびマルチラインコメントが書けます:

- インラインコメント の 例:

```
For /* インラインコメント */ ($vCounter;1;100)  
  ...  
End for
```

- マルチラインコメント は複数行にわたるコメントのことです。この形式のコメントは入れ子にすることができ、4D コードエディターではこれを展開したり 折りたたんだりすることができます。例:

```
For ($vCounter;1;100)
/*
コメント
/*
 詳細なコメント
*/
*/
...
End for
```

演算子

演算子とは、値のチェック・変更・結合に使用する記号または記号のグループです。日常的に使用されている演算子も多くあります。例えば、`1 + 2` という式は加算演算子（プラス記号）を使用し、2つの数値を足し合わせて、3という結果を返します。`=` や `>` などの比較演算子は、2つ以上の値を比較するためのものです。

4Dランゲージでサポートされている演算子は、C や JavaScript など他の言語でも使用されています。ただし、等号比較演算子（`=`）との誤用を防ぐため、代入演算子は `:=` となっています。や、比較演算子（`=`、`>`、`>=`）などの [基本演算子](#) は、数値のほか、ブール、テキスト、日付、時間、ポインター、ピクチャーのデータ型にも使用可能です。JavaScript と同様に、4Dランゲージも [truthy \(真的\)](#) と [falsy \(偽的\)](#) の概念をサポートしており、[短絡演算子](#) で使用されています。

用語

4Dランゲージでは、二項演算子 および 三項演算子 をサポートしています：

- 二項演算子とは、2つの対象に対して演算をおこない、その 2つの対象の間に表示されます（例：`2 + 3`）。
- 三項演算子は 3つの対象に対して演算をおこないます。C と同様、4D の三項演算子は 1つしかありません：三項条件演算子（`a ? b : c`）です。

演算子が影響を与える対象はオペランド（被演算子）と呼ばれます。`1 + 2` という式では、`+` 記号は二項演算子であり、その 2つのオペランドは値 1 と 2 です。

代入

代入演算子（`a := b`）は、`a` の値を `b` の値で初期化、または更新します。

```
$myNumber:=3 // MyNumber 変数に 3 を代入します
$myDate:=!2018/01/21! // 日付リテラルを代入します
$myLength:=Length("Acme") // コマンドの結果 (4) を $myLength に代入します
$col:=New collection // $col を空のコレクションで初期化します
```

代入演算子 `:=` と等号比較演算子 `=` を混同しないように注意してください。`=` とは異なる代入演算子が採用されたのは意図的なことで、他のプログラミング言語で `==` や `====` の使用によって度々起こる間違を避けるためです。このような間違はコンパイラーにとっても発見しにくく、時間を消耗するトラブルシューティングのもとです。

基本演算子

演算の結果は、オペランドの データ型 に依存します。4D はスカラーデータ型に対して様々な演算子をサポートしています。詳細は、各データ型の項にて説明されています：

- [論理演算子](#)（ブール 式に使用）
- [日付演算子](#)
- [時間演算子](#)
- [数値演算子](#)
- [ビットワイズ演算子](#)（倍長整数 式に使用）
- [ピクチャー演算子](#)
- [ポインター演算子](#)
- [文字列演算子](#)

複合代入演算子

4Dでは、代入と演算を組み合わせた 複合代入演算子 をサポートしています。その一例として、加算代入演算子（`+=`）があります。

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

次の複合代入演算子がサポートされています:

演算子	シンタックス	代入される型	例題
加算 (足し算)	Text += Text	テキスト	\$t+=" World" // \$t:=\$t+" World"
	Number += Number	数値	\$n+=5 // \$n:=\$n+5
	Date += Number	日付	\$d+=5 // \$d:=\$d+5
Time += Time	Time += Time	時間	\$t1+\$t2 // \$t1:=\$t1+\$t2
	Time += Number	数値	\$t1+=5 // \$t1:=\$t1+5
	Picture += Picture	ピクチャー	\$p1+\$p2 // \$p1:=\$p1+\$p2 (\$p1 の右に \$p2 を追加します)
Picture += Number	Picture += Number	ピクチャー	\$p1+=5 // \$p1:=\$p1+5 (\$p1 を 5ピクセル右に移動します)
	Number -= Number	数値	\$n-=5 // \$n:=\$n-5
	Date -= Number	日付	\$d-=5 // \$d:=\$d-5
Time -= Time	Time -= Time	時間	\$t1-\$t2 // \$t1:=\$t1-\$t2
	Time -= Number	数値	\$t1-=5 // \$t1:=\$t1-5
	Picture -= Number	ピクチャー	\$p1-=5 // \$p1:=\$p1-5 (\$p1 を 5ピクセル左に移動します)
除算 (割り算)	Number /= Number	数値	\$n/=5 // \$n:=\$n/5
	Time /= Time	時間	\$t1/\$t2 // \$t1:=\$t1/\$t2
	Time /= Number	数値	\$t1/=5 // \$t1:=\$t1/5
Picture /= Picture	Picture /= Picture	ピクチャー	\$p1/\$p2 // \$p1:=\$p1/\$p2 (\$p1 の下に \$p2 を追加します)
	Picture /= Number	ピクチャー	\$p1/=5 // \$p1:=\$p1/5 (\$p1 を 5ピクセル垂直に移動します)
	Text *= Number	テキスト	\$t*="abc" // \$t:=\$t*"abc"
乗算 (かけ算)	Number *= Number	数値	\$n*=5 // \$n:=\$n*5
	Time *= Time	時間	\$t1*\$t2 // \$t1:=\$t1*\$t2
	Time *= Number	数値	\$t1*=5 // \$t1:=\$t1*5
Picture *= Number	Picture *= Number	ピクチャー	\$p1*=5 // \$p1:=\$p1*5 (\$p1 を 5倍にリサイズします)

これらの演算子は、あらゆる [代入可能な式](#) に適用できます (オブジェクトのプロパティやコレクション要素としてのピクチャーを除く)。

"代入先 複合代入演算子 値" と "代入先 := 代入先 演算子 値" は、厳密には等価ではありません。なぜなら、前者の場合、代入先 (変数・フィールド・オブジェクトプロパティ・コレクション要素) は一度しか評価されないからです。たとえば、`getPointer()->+=1` のような式では、`getPointer` メソッドは一度だけ呼び出されます。

[テキストの文字インデックス](#) および [BLOB のバイトインデックス](#) では、これらの演算子はサポートされません。

例題

```

// 加算
$x:=2
$x+=5 // $x=7

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

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

// 減算
$x1:=10
$x1-=5 // $x1=5

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

// 除算
$x3:=10
$x3/=2 // $x3=5

// 乗算
$x2:=10
$x2*=5 // $x2=50

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

```

短絡演算子

演算子 `&&` と `||` は、短絡演算子です。短絡演算子とは、必ずしもすべてのオペランドを評価しない演算子のことです。

[& や | 論理演算子](#) 異なる点は、短絡演算子の `&&` と `||` はブール値を返さないことです。これらは式を `truthy` (真的) または `falsy` (偽的) で評価し、どちらかの式を返します。

AND 短絡演算子 (`&&`)

ルールは以下の通りです。

`Expr1 && Expr2` において:

AND短絡演算子はオペランドを左から右へ評価し、`falsy` と評価された最初のオペランドの値を直ちに返します。すべての値が `truthy` であれば、最後のオペランドの値が返されます。

次の表は、`&&` 演算子の様々なケースをまとめたものです:

Expr1	Expr2	返される値
truthy	truthy	Expr2
truthy	falsy	Expr2
falsy	truthy	Expr1
falsy	falsy	Expr1

例題 1

```

var $v : Variant

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

```

例題 2

オンラインストアで、税率が適用される商品とされない商品があるとします。

税金を計算するには、価格に税率をかけますが、税率は指定されていない場合があります。

そこで、次のように書くことができます:

```

var $tax : Variant

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

```

`taxRate` が `NULL` (または未定義) の場合、`$tax` は `NULL` となり、それ以外の場合には計算結果が格納されます。

例題 3

短絡演算子は、次のようなテストに有効です:

```

If(( $myObject#Null ) && ( $myObject.value > 10 ))
    // コード
End if

```

もし `$myObject` が `Null` であれば、第2引数は実行されないため、エラーは発生しません。

OR 短絡演算子 (||)

`||` 演算子は、指定されたオペランドのうち 1つの値を返します。式は左から右に評価され、以下のルールに基づいて "短絡" 評価の可能性をテストされます。

`Expr1 || Expr2` において:

`Expr1` が `truthy` であれば、`Expr2` は評価されず、計算は `Expr1` を返します。

`Expr1` が `falsy` の場合、計算は `Expr2` を返します。

次の表は、`||` 演算子の様々なケースと返される値をまとめたものです:

Expr1	Expr2	返される値
truthy	truthy	Expr1
truthy	falsy	Expr1
falsy	truthy	Expr2
falsy	falsy	Expr2

例題 1

`Employee` というテーブルがあるとします。従業員には電話番号を入力している人と入力していない人がいます。つまり、`$emp.phone` は `NULL` である可能性があり、テキスト変数に `NULL` を代入することはできません。そこで、次のように書くことができます:

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

この場合、`$phone` には電話番号か、"n/a" という文字列のどちらかが格納されます。

例題 2

`name` フィールドと、既婚女性のための `maiden name` (旧姓) フィールドを持つ Person テーブルがあるとします。

次の例は、旧姓データがあれば変数に格納し、なければその人の名前を変数に格納します。

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

優先順位

演算子 `&&` と `||` は、論理演算子 `&` および `|` と同じ優先順位を持ち、左から右へ評価されます。

つまり、`a || b && c` は、`(a || b) && c` として評価されます。

三項演算子

三項演算子を使うと、条件式を 1行で書くことができます。たとえば、`If...Else` 文を完全に置き換えることができます。

三項演算子は 3つのオペランドを次の順序で受け取ります:

- 条件とクエスチョンマーク (?)
- 条件が `truthy` である場合に実行される式、その後にコロン (:)
- 条件が `falsy` の場合に実行される式

シンタックス

シンタックスは次のとおりです:

```
条件 ? truthy時の式 : falsy時の式
```

トーカンシンタックス にはコロンが使われているため、競合を避けるには、コロン `:` の後にスペースを入れる、または、トーカンは括弧でくることが推奨されます。

例題

単純な例

```
var $age : Integer  
var $beverage : Text  
  
$age:=26  
$beverage:=($age>=20) ? "ビール" : "ジュース"  
  
ALERT($beverage) // "ビール"
```

テーブルのデータを扱う例

この例では、人のフルネームを変数に格納し、ファーストネームやラストネームが指定されていないケースに対応します:

```
var $fullname : Text

// どちらか片方の情報が欠けている場合には存在する方を格納し、両方存在しない場合は空の文字列を格納します。
$fullname:=($person.firstname && $person.lastname) ? ($person.firstname+" "+$person.lastname) : ($person
```

Truthy と Falsy

各値はデータ型のほかに、固有のブール値を持ちます。このブール値は **truthy** (真的) または **falsy** (偽的) です。

truthy および falsy の値は [短絡演算子](#) および [三項演算子](#) の場合にのみ評価されます。

以下の値は falsy です:

- false
- Null
- 未定義
- Null オブジェクト
- Null コレクション
- Null ポインター
- Null ピクチャー
- Null 日付 !00-00-00!
- "" - 空の文字列
- [] - 空のコレクション
- {} - 空のオブジェクト

上記以外の値はすべて truthy と評価されます。

- 0 - 数値のゼロ (整数かどうかを問わず)

4Dでは、truthy と falsy の評価は値の 使用性 を反映します。つまり、truthy な値は存在し、エラーや予期せぬ結果を発生させずにコードによって処理できることを意味します。その目的は、オブジェクトやコレクションにおける `undefined` や `null` 値を扱うための便利な方法を提供し、実行時エラーを回避するために必要な [If...Else](#) 文の数を少なくすることにあります。

たとえば、[OR 短絡演算子](#) を使用すると:

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

`$object` が `value` プロパティを含まない場合、または同プロパティが `null` の場合に、デフォルト値が代入されます。つまり、この演算子は特定の値ではなく、その値の存在や使用性をチェックするのです。なお、数値の 0 は存在しており使用可能であるため、特別に扱われる事はなく、truthy です。

コレクション、オブジェクト、文字列を表す値については、"空" の値は falsy とみなされます。これは、空の値に遭遇したときに、デフォルト値を割り当てたい場合に便利です。

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

データタイプの概要

4Dにおいてデータは、主にデータベースフィールドと 4D ランゲージという2つの場所で、そのタイプに応じて扱われます。

この2つはおよそ同じものですが、データベースレベルで提供されているいくつかのデータタイプはランゲージにおいては直接利用可能ではなく、自動的に適宜変換されます。同様に、いくつかのデータタイプはランゲージでしか利用できません。各場所で利用可能なデータタイプと、ランゲージでの宣言の仕方の一覧です：

データタイプ	データベース	ランゲージ	var 宣言	C_ または ARRAY 宣言
文字列	○	テキストに変換	-	-
テキスト	○	○	Text	C_TEXT , ARRAY TEXT
日付	○	○	Date	C_DATE , ARRAY DATE
時間	○	○	Time	C_TIME , ARRAY TIME
ブール	○	○	Boolean	C_BOOLEAN , ARRAY BOOLEAN
整数	○	倍長整数に変換	Integer	ARRAY INTEGER
倍長整数	○	○	Integer	C_LONGINT , ARRAY LONGINT
64ビット整数	○ (SQL)	実数に変換	-	-
実数	○	○	Real	C_REAL , ARRAY REAL
未定義	-	○	-	-
Null	-	○	-	-
ポインター	-	○	Pointer	C_POINTER , ARRAY POINTER
ピクチャー	○	○	Picture	C_PICTURE , ARRAY PICTURE
BLOB	○	○	Blob , 4D.Blob	C_BLOB , ARRAY BLOB
オブジェクト	○	○	Object	C_OBJECT , ARRAY OBJECT
コレクション	-	○	Collection	C_COLLECTION
バリアント(2)	-	○	Variant	C_VARIANT

(1) ORDA では、オブジェクト (エンティティ) を介してデータベースフィールドを扱うため、オブジェクトにおいて利用可能なデータタイプのみがサポートされます。詳細については [オブジェクト](#) のデータタイプの説明を参照ください。

(2) バリアントは実際のところデータ タイプではなく、あらゆるデータタイプの値を格納することのできる 変数 タイプです。

デフォルト値

コンパイラー指示子によって変数の型が決まるとき、変数はデフォルトの値を受け取り、割り当てがされない限りセッションの間はその値を保ち続けます。

デフォルト値は変数の型に依存します：

タイプ	デフォルト値
ブール	False
日付	00-00-00
倍長整数	0
時間	00:00:00
ピクチャー	ピクチャーサイズ=0
実数	0
ポインター	Nil=true
テキスト	""
BLOB	Blob サイズ=0
オブジェクト	null
コレクション	null
バリアント	未定義

データタイプの変換

4D ランゲージには、データタイプ間の変換をおこなう演算子やコマンドがあります。4D ランゲージはデータタイプをチェックします。たとえば、"abc"+0.5+!12/25/96!-?00:30:45?のように記述することはできません。これは、シンタックス (構文) エラーになります。

次の表は、基本のデータタイプ、変換できるデータタイプ、それを実行する際に使用するコマンドを示しています:

データタイプ	文字列に変換	数値に変換	日付に変換	時間に変換	ブールに変換
文字列 (1)		Num	Date	Time	Bool
数値 (2)	String				Bool
日付	String				Bool
時間	String				Bool
ブール		Num			

(1) JSON形式の文字列は `JSON Parse` コマンドを使ってスカラーデータ、オブジェクト、あるいはコレクションに変換することができます。

(2) 時間は数値として扱うことができます。

注: この表に示すデータ変換の他に、演算子と他のコマンドを組み合せることで、より洗練されたデータ変換を実行することができます。

BLOB

BLOB (Binary Large OBject) フィールド・変数・式とは、連続した可変長バイトであり、各バイトを個々にアドレス指定可能な 1つのまとまったオブジェクトとして取り扱うことができます。

BLOB は全体がメモリにロードされます。BLOB変数はメモリ内にだけ保持され、存在します。BLOBフィールドは、レコードの他フィールドと同様に、ディスクからメモリにロードされます。

大量のデータを保持できる他のフィールドタイプ (ピクチャーなど) と同様に、レコードを更新しても BLOBフィールドはメモリに複製されません。したがって、`Old` および `Modified` コマンドを BLOBフィールドに適用しても、返される結果は意味を持ちません。

BLOB の種類

4Dランゲージでは、BLOB を扱う方法が 2つあります：

- スカラー値として： BLOB は BLOB変数またはフィールドに格納され、変更することができます。
- オブジェクト (`4D.Blob`) として：`4D.Blob` は BLOBオブジェクトです。オリジナルの BLOB を変更することなく、BLOBそのもの、またはその一部を `4D.Blob` に格納できます。この方法を **ボクシング** と呼びます。`4D.Blob` をインスタンス化する方法については、[Blobクラス](#) を参照ください。

各 BLOBタイプには、それぞれ利点があります。次の表を参考にして、どちらがニーズに合うかを確認してください：

	BLOB	4D.Blob
変更可能	○	×
オブジェクトやコレクション内で共有可能	×	○
参照渡し*	×	○
バイトにアクセスする際のパフォーマンス	+	-
最大サイズ	2GB	メモリ

*スカラーBLOB をパラメーターとして受け取るように設計された 4Dコマンドとは異なり、スカラーBLOB をメソッドに渡すと、メモリ内で複製されます。メソッドを使用する場合は、参照によって渡される BLOBオブジェクト (`4D.Blob`) を使用する方が効率的です。

デフォルトで、4D はスカラーBLOB の最大サイズを 2GB に設定していますが、OSや空き容量によっては、この制限サイズが小さくなる場合があります。

BLOB に演算子を適用することはできません。

変数がスカラーBLOB と `4D.Blob` のどちらを格納しているかの確認

値が BLOB型またはオブジェクト型であるかどうかを確認するには、[Value type](#) コマンドを使用します。特定のオブジェクトが BLOBオブジェクト (`4D.Blob`) であることを確認するには、[OB instance of](#) を使用します。

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

$type:= Value type($myblobObject) // 38 (オブジェクト)
$is4DBlob:= OB Instance of($myblobObject; 4D.Blob) // true
```

BLOB を引数として渡す

スカラーBLOB や BLOBオブジェクトは、4Dコマンドまたは 4Dプラグインの引数として渡すことができます。

BLOB および BLOBオブジェクトの 4Dコマンドへの受け渡し

BLOB を引数として受け取る 4Dコマンドには、スカラーBLOB または `4D.Blob` を渡すことができます:

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

4Dコマンドの中には、元の BLOB を変更するものがあり、これらは `4D.Blob` タイプをサポートしていません:

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

BLOB および BLOBオブジェクトのメソッドへの受け渡し

BLOB や BLOBオブジェクト (`4D.Blob`) は、メソッドに渡すことができます。参照で渡される BLOBオブジェクトとは異なり、スカラーBLOB はメソッドに渡されるとメモリ内で複製されることに注意してください。

ポインターを使ったスカラーBLOB の参照渡し

スカラーBLOB をメモリ上に複製することなくメソッドに渡すには、ポインターを使用します。その場合は BLOB変数へのポインターを定義し、そのポインターを引数として渡します。

例:

```
// BLOB型の変数を定義します  
var $myBlobVar: Blob  
// 4Dコマンドに引数として BLOB を渡します  
SET BLOB SIZE($myBlobVar;1024*1024)
```

```
// 外部ルーチンに BLOB を引数として渡します  
$errCode:=Do Something With This blob($myBlobVar)
```

```
// BLOB を引数としてメソッドに渡し、戻り値を BLOB で受け取ります  
var $retrieveBlob: Blob  
retrieveBlob:=Fill_Blob($myBlobVar)
```

```
// BLOB のポインターをメソッドに渡します  
COMPUTE BLOB(>$myBlobVar)
```

プラグイン開発にあたっての注意: BLOB 引数は “&O” (数字の0ではなく、アルファベットの”O”) として宣言します。

BLOB変数の代入

BLOB変数は相互に代入することができます:

例:

```
// BLOB型の変数を二つ宣言します
var $vBlobA; $vBlobB : Blob
// 一つ目の BLOB に10K のサイズを割り当てます
SET BLOB SIZE($vBlobA;10*1024)
// 一つ目の BLOB を二つ目の BLOB に代入します
$vBlobB:=$vBlobA
```

BLOB の自動変換

スカラーBLOB が BLOBオブジェクトに (またはその逆) 割り当てられると、4Dはそれらを自動的に変換します。たとえば:

```
// BLOB型の変数とオブジェクト変数を作成します
var $myBlob: Blob
var $myObject : Object

// $myObject の "blob" というプロパティに BLOB変数を代入します
$myObject:=New object("blob"; $myBlob)

// $myBlob に格納される BLOB は自動的に 4D.Blob に変換されます
$type:= OB Instance of($myObject.blob; 4D.Blob) // true

// 4D.Blob から スカラーBLOB への変換
$myBlob:= $myObject.blob
$type:= Value type($myBlob) // BLOB
```

4D.Blob をスカラーBLOB に変換する際に、4D.Blob のサイズがスカラーBLOB の最大サイズを超える場合、結果のスカラーBLOB は空になります。たとえば、スカラーBLOB の最大サイズが 2GB の場合、2.5GB の 4D.Blob をスカラーBLOB に変換すると、空の BLOB が得られます。

スカラーBLOB の変更

BLOBオブジェクトとは異なり、スカラーBLOB は変更することができます。たとえば:

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

BLOB内バイトへの個別アクセス

スカラーBLOB のバイトへのアクセス

中カッコ {...} を使用し、BLOB の各バイトを個別にアクセスすることができます。BLOB 内では、各バイトに 0 から N-1 の番号が割り当てられています。N は BLOB のサイズです:

```
// BLOB型変数を定義します
var $vBlob : Blob
// BLOB のサイズを 256バイトに設定します
SET BLOB SIZE($vBlob;256)
// 次のループは、BLOB の各バイトをゼロに初期化します
For(vByte;0;BLOB size($vBlob)-1)
    $vBlob{vByte}:=0
End for
```

BLOB の各バイトはすべて個別にアドレス指定できるため、BLOB変数またはフィールドには何でも格納できます。

4D.Blob のバイトへのアクセス

大カッコ [...] を使用し、4D.Blob の各バイトを個別にアクセスすることができます。

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

4D.Blob は変更できないため、このシンタクスで 4D.Blob のバイトを読むことはできますが、変更することはできません。

ブール

ブールのフィールド、変数、式は、true（真）またはfalse（偽）のいずれかになります。

ブール関数

4Dにはブール演算に使用することのできる、ブール関数があります: `True`, `False`, `Not`。詳細については、これらのコマンドの説明を参照ください。

例題

ボタンの値に基づいて、ブール変数に値を設定します。myButton ボタンがクリックされたら myBoolean に true を、クリックされていなければ false を設定します。ボタンがクリックされるとボタン変数の値は1になります。

```
If(myButton=1) // ボタンがクリックされたら  
    myBoolean:=True // myBoolean を true に設定  
Else // ボタンがクリックされていなければ  
    myBoolean:=False // myBoolean を false に設定  
End if
```

上のコードは以下のように一行で書くこともできます。

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

論理演算子

4Dは、ブール式に対して機能する次の論理演算子をサポートしています: 論理積 (AND) と論理和 (OR)。論理積 (AND) は両方の式が true である場合に true を返します。論理和 (OR) は少なくとも一方の式が true の時に true を返します。次の表に、論理演算子を示します:

演算子	シンタックス	戻り値	式	値
AND	ブール & ブール	ブール	("A" = "A") & (15 # 3)	True
			("A" = "B") & (15 # 3)	False
			("A" = "B") & (15 = 3)	False
OR	ブール ブール	ブール	("A" = "A") (15 # 3)	True
			("A" = "B") (15 # 3)	True
			("A" = "B") (15 = 3)	False

論理演算子 (AND) の真偽表を示します:

Expr1	Expr2	Expr1 & Expr2
True	True	True
True	False	False
False	True	False
False	False	False

論理演算子 (OR) の真偽表を示します:

Expr1	Expr2	Expr1 Expr2
True	True	True
True	False	True
False	True	True
False	False	False

Tip: 式1と式2の排他的結合子演算を実行する必要がある場合、次の評価式を使用します:

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

4Dランゲージはブールのコンテキストにおいて、短絡演算子（`&&` と `||`）および `Truthy` (真的) と `Falsy` (偽的) の概念もサポートしています。

コレクション

コレクションとは、類似または混在した型（テキスト、数値、日付、オブジェクト、布尔、コレクション、null）の値が順番に並べられたリストです。

コレクション型の変数を扱うには、オブジェクト記法を使用します（[オブジェクト記法の使用](#) 参照）。

コレクション要素にアクセスするには、大カッコ内に要素番号を渡します：

```
collectionRef[expression]
```

expression には正の整数を返す有効な 4D式であればどんなものでも渡すことができます。例：

```
myCollection[5] // コレクションの6番目の要素にアクセス  
myCollection[$var]
```

注：コレクション要素は0 番から始まるということに注意してください。

コレクションの要素に値を代入したり、コレクション要素の値を取得したりすることができます：

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

コレクションの最後の要素を超える要素番号（インデックス）を指定した場合、コレクションは自動的にリサイズされ、途中のすべての値には null 値が割り当てられます：

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

初期化

`New collection` コマンドを使うなどして、コレクションはあらかじめ初期化しておく必要があります。初期化しない場合、要素の取得や変更はシングルスラッシュとなります。

例：

```
var $colVar : Collection // コレクション型の 4D変数の宣言  
$colVar:=New collection // コレクションの初期化と 4D変数への代入
```

通常コレクションと共有コレクション

二種類のコレクションを作成することができます：

- `New collection` コマンドを使用して作成する通常（非共有）コレクション。通常のコレクションは特別なアクセスコントロールをせずに編集可能ですが、プロセス間で共有することはできません。
- `New shared collection` コマンドを使用して作成する共有コレクション。共有コレクションはプロセス間（プリエンティブ・スレッド含む）で共有可能なコレクションです。共有コレクションへのアクセスは `Use...End use` 構造によって管理されています。

詳細な情報については、[共有オブジェクトと共有コレクション](#) を参照ください。

コレクション関数

4D コレクションへの参照は、コレクションの メンバー関数 と呼ばれる特別なクラス関数を利用することができます。コレクション関数は [クラス API リファレンス](#) にまとめられています。

たとえば:

```
$newCol:=$col.copy() // $col を $newCol にディープ・コピー  
$col.push(10;100) // 10 と 100 をコレクションに追加
```

一部の関数は元のコレクションを変更して返すので、つぎのように連続して呼び出すことが可能です:

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

propertyPath 引数

いくつかのコレクション関数は引数として *propertyPath* を受け入れます。この引数は以下のように用いることができます:

- オブジェクトプロパティ名、例えば "lastName"
- オブジェクトプロパティパス (ドット文字で繋げられたサブプロパティの階層シーケンスなど)。例: "employee.children.firstName"

警告: 関数に *propertyPath* 引数を渡す場合、そのプロパティ名には "." (ドット)、"[]" (大カッコ)、あるいは " " (スペース) を使えません。これらを使用するとパスを正しく解析できなくなります:

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

日付

日付フィールド、変数、式として認識できる範囲は、100/1/1 から 32,767/12/31 までです。(日本語版の 4D を使用した場合、日付の順序は年/月/日の順になります。)

C_DATE によって宣言された日付は 32767 年までの範囲に対応していますが、システムを経由する処理によっては上限にさらなる制限が課せられます。

注: 4D ランゲージリファレンスでは、コマンド説明における日付引数はとくに明記されていない限り、「日付」と表記されています。

日付リテラル

日付リテラル定数は、エクスクラメーションマーク (! ... !) で囲んで表します。日付は ISOフォーマット (!YYYY-MM-DD!) を使って記述します。下記に、日付定数の例を示します:

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

空の日付は、!00-00-00! のように指定します。

Tip: メソッドエディターでは空の日付を入力するためのショートカットが提供されています。空の日付を入力するには、エクスクラメーションマーク (!) の入力後に Enterキーを押します。

注:

- 互換性の理由から、4D は二桁の年次の入力を受け付けます。数字が 30 以上の場合は 20世紀 (1900年代)、30未満の場合は 21世紀 (2000年代) であると認識します (ただしデフォルト設定が SET DEFAULT CENTURY コマンドを使用して変更されていない場合に限ります)。
- "地域特有のシステム設定を使う" オプション ([メソッドページ](#) 参照) にチェックがされている場合、システムで定義されている日付フォーマットを使用する必要があります。一般的に、US環境においては、日付は月/日/年の形式で入力され、値はスラッシュ "/" で区切られます。

日付演算子

演算子	シンタックス	戻り値	式	値
日付の差	日付 - 日付	数値	$!2017-01-20! - !2017-01-01!$	19
日付の加算	日付 + 数値	日付	$!2017-01-20! + 9$	$!2017-01-29!$
日付の減算	日付 - 数値	日付	$!2017-01-20! - 9$	$!2017-01-11!$
等しい	日付 = 日付	ブール	$!2017-01-01! == !2017-01-01!$	True
			$!2017-01-20! == !2017-01-01!$	False
異なる	日付 # 日付	ブール	$!2017-01-20! \# !2017-01-01!$	True
			$!2017-01-20! \# !2017-01-20!$	False
大きい	日付 > 日付	ブール	$!2017-01-20! > !2017-01-01!$	True
			$!2017-01-20! > !2017-01-20!$	False
小さい	日付 < 日付	ブール	$!2017-01-01! < !2017-01-20!$	True
			$!2017-01-20! < !2017-01-20!$	False
以上	日付 >= 日付	ブール	$!2017-01-20! >= !2017-01-01!$	True
			$!2017-01-01! >= !2017-01-20!$	False
以下	日付 <= 日付	ブール	$!2017-01-01! <= !2017-01-20!$	True
			$!2017-01-20! <= !2017-01-01!$	False

Null と 未定義

Null および未定義は、式の値が未知のケースを扱うデータ型です。

Null

Null は null の値のみをとることのできる特殊なデータタイプです。この値は、値を持たない式によって返されます。

4D ランゲージやオブジェクトフィールド属性においては、 Null 関数を使ってnull値を扱います。この関数をつぎの式と組み合わせて使うことで、null値の設定や比較をおこなうことができます：

- オブジェクトの属性
- コレクションの要素
- オブジェクト型、コレクション型、ポインター型、ピクチャ型、バリアント型の変数

未定義

未定義 (undefined) は、実際にはデータタイプではありません。未定義は、まだ定義されていない変数を示します。関数（結果を返すプロジェクトメソッド）は、メソッド内で戻り値 (\$0) に未定義式が代入されている場合、未定義値を返すことがあります。未定義式とは、未定義の変数を一つ以上使っている式のことです。フィールドは、未定義にはできません（フィールドの場合、 Undefined コマンドは常に False を返します）。バリアント型変数は undefined がデフォルト値となっています。

例題

オブジェクトプロパティを対象に、 Undefined および Null コマンドを使用した場合の結果の例です：

```
C_OBJECT($vEmp)
$vEmp:=New object
$vEmp.name:="Smith"
$vEmp.children:=Null

$undefined:=Undefined($vEmp.name) // false
>null:=( $vEmp.name=Null) //false

$undefined:=Undefined($vEmp.children) // false
>null:=( $vEmp.children=NULL) //true

$undefined:=Undefined($vEmp.parent) // true
>null:=( $vEmp.parent=NULL) //true
```

数値 (実数、倍長整数、整数)

数値とは、以下を示す総称です：

- 実数のフィールド、変数、または式。実数データタイプの範囲は、 $\pm 1.7e\pm 308$ (有効数字13桁) です。
- 倍長整数のフィールド、変数、または式。倍長整数 (4バイト整数) データタイプの範囲は、 $-2^{31}..(2^{31}-1)$ です。
- 整数のフィールド、変数、または式。整数 (2バイト整数) データタイプの範囲は、 $-32,768..32,767$ ($2^{15}..(2^{15}-1)$) です。

注：整数フィールドの値は、4D ランゲージで使用される際には自動的に倍長整数に変換されます。

数値データタイプは、異なる数値データタイプに代入することができます。このとき、4Dが必要に応じて変換、切り捨て、丸め処理をおこないます。ただし、値が範囲外の場合には、変換は正しい値を返しません。数値データタイプは式の中に混在させて使用することができます。

注：4D ランゲージリファレンスでは、実際のデータタイプに関わらず、コマンド説明における実数、整数、倍長整数の引数はとくに明記されていない限り、数値と表記されています。

数値リテラル

数値リテラル定数は、実数として記述します。下記に数値定数の例をいくつか示します：

27
123.76
0.0076

デフォルトの小数点はシステム言語に関係なくピリオド (.) です。"地域特有のシステム設定を使う" オプション ([メソッドページ 参照](#)) にチェックがされている場合、システムで定義されている小数点を使用する必要があります。

負の数値は、マイナス記号 (-) を付けて指定します。たとえば：

-27
-123.76
-0.0076

数値演算子

演算子	シンタックス	戻り値	式	値
加算 (足し算)	数値 + 数値	数値	2 + 3	5
減算 (引き算)	数値 - 数値	数値	3 - 2	1
乗算 (かけ算)	数値 * 数値	数値	5 * 2	10
除算 (割り算)	数値 / 数値	数値	5 / 2	2.5
倍長整数を返す除算	数値 ¥ 数値	数値	5 ¥ 2	2
モジューロ	数値 % 数値	数値	5 % 2	1
指数	数値 ^ 数値	数値	2 ^ 3	8
等しい	数値 = 数値	ブール	10 = 10	True
			10 = 11	False
異なる	数値 # 数値	ブール	10 # 11	True
			10 # 10	False
大きい	数値 > 数値	ブール	11 > 10	True
			10 > 11	False
小さい	数値 < 数値	ブール	10 < 11	True
			11 < 10	False
以上	数値 >= 数値	ブール	11 >= 10	True
			10 >= 11	False
以下	数値 <= 数値	ブール	10 <= 11	True
			11 <= 10	False

モジューロ演算子 % は最初の数値を 2番目の数値で除算し、その余りの整数を返します。次に例を示します:

- 10 % 2は、0を返します。10 は 2 で割り切れるからです。
- 10 % 3は、1を返します。余りが 1 だからです。
- 10.5 % 2は、0を返します。余りが整数ではない (0.25) からです。

警告:

- モジューロ演算子 % は倍長整数の範囲内 (-2^31 から (2^31)-1 まで) の数値に対して有効な値を返します。この範囲外の数値のモジューロ演算を実行するには、Mod コマンドを使用します。
- 倍長整数を返す除算演算子 ¥ は、整数値の有効値を返します。

優先順位

式を評価する順番を優先順位と呼びます。4D における優先順位は厳密に左から右で、代数的順序は採用されていません。たとえば:

3+4*5

これは 35 を返します。最初に式 3+4 の結果 7 を求め、それに 5 を乗じるので、結果は 35 になります。

左から右の優先順位を変更するには、必ずカッコを使用します。たとえば:

3+(4*5)

この式は、23 を返します。カッコがあるため、最初に式 (4*5) の結果 20 を求め、それに 3 を加えて、結果は 23 になります。

カッコは、他のカッコの組の内側にネストすることができます。式の評価が正しくおこなわれるよう、必ず各左カッコに対応する右カッコを指定してください。カッコの不足または誤用は、予測できない結果や、式の無効化につながります。またコンパイルする場合は、左カッコと右カッコは同じ数でなければなりません。

せん。組になっていないカッコはシンタックスエラーとして検出されます。

ビットワイズ演算子

ビットワイズ演算子は、倍長整数式や値に対して演算をおこないます。

ビットワイズ演算子に整数値または実数値を渡すと、4Dは値を倍長整数値として評価してから、ビットワイズ演算子を使用した式を計算します。

ビットワイズ演算子を使用する場合、倍長整数値を32ビットの配列と考える必要があります。これらのビットには、右から左に0~31の番号が付けられます。

それぞれのビットは0か1なので、倍長整数値は32のブール値を格納できる値を考えることもできます。1に等しいビットは true、0に等しいビットは false を意味します。

ビットワイズ演算子を使用する式は倍長整数値を返します。Bit Test 演算子の場合、式は例外的にブール値を返します。次の表にビットワイズ演算子とそのシンタックスを示します：

演算子	演算子	シンタックス	戻り値
Bitwise AND	&	Long & Long	Long
Bitwise OR (inclusive)		Long Long	Long
Bitwise OR (exclusive)	^	Long ^ Long	Long
Left Bit Shift	<<	Long << Long	Long (注記1 参照)
Right Bit Shift	>>	Long >> Long	Long (注記1 参照)
Bit Set	?+	Long ?+ Long	Long (注記2 参照)
Bit Clear	?-	Long ?- Long	Long (注記2 参照)
Bit Test	??	Long ?? Long	Boolean (注記2 参照)

注記

- Left Bit Shift および Right Bit Shift 演算では、2番目のオペランドは、結果値において1番目のオペランドのビットがシフトされるビット数を示します。したがって、この2番目のオペランドは、0~31の間でなければなりません。0ビットシフトするとその値がそのまま返されます。また、31ビットより多くシフトするとすべてのビットがなくなるので、0x00000000が返されます。それ以外の値を2番目のオペランドとして渡した場合、結果は意味のない値になります。
- Bit Set、Bit Clear、Bit Test 演算では、2番目のオペランドは、作用の対象となるビット番号を示します。したがって、この2番目のオペランドは0 ~ 31の間です。そうでない場合、式の結果は意味のないものになります。

次の表は、ビットワイズ演算子とその効果を示します：

演算子	説明
Bitwise AND	<p>それぞれの結果ビットは2つのオペランドのビットの論理ANDです。</p> <p>下記は、論理ANDの真偽表です:</p> <ul style="list-style-type: none"> • $1 \& 1 \rightarrow 1$ • $0 \& 1 \rightarrow 0$ • $1 \& 0 \rightarrow 0$ • $0 \& 0 \rightarrow 0$ <p>すなわち、両オペランドのビットが 1 の場合、結果ビットが 1 になり、その他の場合は結果ビットが 0 になります。</p>
Bitwise OR (inclusive)	<p>それぞれの結果ビットは2つのオペランドのビットの論理ORです。</p> <p>下記は、論理ORの真偽表です:</p> <ul style="list-style-type: none"> • $1 1 \rightarrow 1$ • $0 1 \rightarrow 1$ • $1 0 \rightarrow 1$ • $0 0 \rightarrow 0$ <p>すなわち、いずれかのオペランドのビットが 1 の場合、結果ビットが 1 になり、その他の場合は結果ビットが 0 になります。</p>
Bitwise OR (exclusive)	<p>それぞれの結果ビットは2つのオペランドのビットの排他的論理ORです。</p> <p>下記は、排他的論理ORの真偽表です:</p> <ul style="list-style-type: none"> • $1 \wedge 1 \rightarrow 0$ • $0 \wedge 1 \rightarrow 1$ • $1 \wedge 0 \rightarrow 1$ • $0 \wedge 0 \rightarrow 0$ <p>すなわち、オペランドのビットのいずれか一方だけが 1 の場合、結果ビットが 1 になり、その他の場合は結果ビットが 0 になります。</p>
Left Bit Shift	<p>最初のオペランド値が結果値に設定され、次に結果ビットが2番目のオペランドで示されたビット数だけ左にシフトします。左側のビットがなくなり、右側の新しいビットは0に設定されます。</p> <p>注記: 正の数だけを考えると、Nビット左にシフトすることは、2^Nを掛けることと同じです。</p>
Right Bit Shift	<p>最初のオペランド値が結果値に設定され、次に結果ビットが2番目のオペランドで示されたビット数だけ右にシフトします。右側のビットがなくなり、左側の新しいビットは0に設定されます。</p> <p>注記: 正の数だけを考えると、Nビット右にシフトすることは、2^Nで割ることと同じです。</p>
Bit Set	最初のオペランド値が結果値に設定され、次に結果ビットのうち2番目のオペランドで示されたビットが1に設定されます。他のビットはそのままです。
Bit Clear	最初のオペランド値が結果値に設定され、次に結果ビットのうち2番目のオペランドで示されたビットが0に設定されます。他のビットはそのままです。
Bit Test	最初のオペランドのうち、2番目のビットで示されたビットが1の場合、trueが返されます。最初のオペランドのうち、2番目のビットで示されたビットが0の場合、falseが返されます。

例題

演算子	例題	戻り値
Bitwise AND	0x0000FFFF & 0xFF00FF00	0x0000FF00
Bitwise OR (inclusive)	0x0000FFFF 0xFF00FF00	0xFF00FFFF
Bitwise OR (exclusive)	0x0000FFFF ^ 0xFF00FF00	0xFF00000F
Left Bit Shift	0x0000FFFF << 8	0x00FFFF00
Right Bit Shift	0x0000FFFF >> 8	0x0000000F
Bit Set	0x00000000 ?+ 16	0x00010000
Bit Clear	0x00010000 ?- 16	0x00000000
Bit Test	0x00010000 ?? 16	True

オブジェクト

オブジェクト型の変数・フィールド・式にはさまざまなデータを格納することができます。4D のネイティブなオブジェクトの構造は、よくある「プロパティ/値」(または「属性/値」というペア (連想配列) に基づいています。これらオブジェクトの記法は JSON をもとにしていますが、完全に同じというわけではありません。

- プロパティ名は必ずテキストで表現されます。プロパティ名には [命名規則](#) があります。
- プロパティ値は以下のどれかの型で表現されます:
 - 数値 (実数、整数、等)
 - テキスト
 - null
 - ブール
 - ポインター (`JSON Stringify` コマンドの使用、またはコピーの際に評価されます)
 - 日付 (日付型あるいは ISO 日付フォーマット文字列)
 - オブジェクト(1) (オブジェクトは入れ子にすることができます)
 - ピクチャー(2)
 - コレクション

(1) [エンティティ](#) や [エンティティセレクション](#) などの ORDA オブジェクトは オブジェクトフィールド には保存することができませんが、メモリ内の オブジェクト変数 に保存することは可能です。

(2) デバッガー内でテキストとして表示したり、JSON へと書き出されたりした場合、ピクチャー型のオブジェクトプロパティは "[object Picture]" と表されます。

警告: 属性名は大文字と小文字を区別するという点に注意してください。

オブジェクト型の変数・フィールド・式を操作するには [オブジェクト記法](#) を用いるか、オブジェクト (ランゲージ) テーマが提供する従来のコマンドを使用します。オブジェクト型フィールドに対して処理をおこなうには `QUERY BY ATTRIBUTE`、`QUERY SELECTION BY ATTRIBUTE` や `ORDER BY ATTRIBUTE` など、クエリ テーマの特定のコマンドも使用することができます。

オブジェクト記法を使ってアクセスされたそれぞれのプロパティ値は式とみなされます。4D 内で式が期待される場所であれば、どこでもこのような値を使用することができます:

- 4D コード内。メソッド (メソッドエディター) に書いても、外部化 (フォーミュラ、`PROCESS 4D TAGS` あるいは Web Server によって処理される 4D tags ファイル、4D Write Pro ドキュメントなど) しても使用可能です。
- デバッガー及びランタイムエクスプローラーの式エリア内。
- フォームエディターにおいて、フォームオブジェクトのプロパティリスト内。変数あるいは式フィールド内の他、様々なセレクションリストボックス及びカラムの式 (データソース、背景色、スタイル、フォントカラー等) において使用可能です。

初期化

`New object` コマンドを使うなどして、オブジェクトはあらかじめ初期化しておく必要があります。初期化しない場合、プロパティ値の取得や変更はシンタックスエラーとなります。

例:

```
C_OBJECT($obVar) // オブジェクト型の変数の作成  
$obVar:=New object // オブジェクトの初期化と変数への代入
```

通常オブジェクトと共有オブジェクト

二種類のオブジェクトを作成することができます:

- `New object` コマンドを使用して作成する通常 (非共有) オブジェクト。通常のオブジェクトは特別なアクセスコントロールをせずに編集可能ですが、プロセス間で共有することはできません。

- `New shared object` コマンドを使用して作成する共有オブジェクト。共有オブジェクトはプロセス間 (プリエンティブ・スレッド含む) で共有可能なオブジェクトです。共有オブジェクトへのアクセスは `Use...End use` 構造によって管理されています。詳細な情報については、[共有オブジェクトと共有コレクション](#) を参照ください。

オブジェクト記法の使用

オブジェクト記法を使うと、トークンのチェーンを通してオブジェクトのプロパティ値にアクセスすることができます。

オブジェクトプロパティ

オブジェクト記法では、オブジェクトプロパティは二通りの方法でアクセスすることができます:

- "ドット"記号を使用する方法: > `object.propertyName`

例:

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

- 大カッコ内の文字列を使用する方法: > `object["propertyName"]`

例:

```
$vName:=employee["name"]
// または:
$property:="name"
$vName:=employee[$property]
```

オブジェクトプロパティ値には、オブジェクトやコレクションも設定することができます。これらのサブプロパティにアクセスするため、オブジェクト記法では連続した字句を受け入れることができます:

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

オブジェクトを格納、あるいは返すあらゆるランゲージ要素に対してオブジェクト記法を使用できます。たとえば:

- オブジェクト自身 (変数、フィールド、オブジェクトプロパティ、オブジェクト配列、コレクション要素などに保存されているもの) 例:

```
$age:=$myObjVar.employee.age // 変数
$addr:=[Emp]data_obj.address // フィールド
$city:=$addr.city // オブジェクトプロパティ
$pop:=$aObjCountries[2].population // オブジェクト配列
$val:=$myCollection[3].subvalue // コレクション要素
```

- オブジェクトを返す 4D コマンド 例:

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

- オブジェクトを返す プロジェクトメソッド 例:

```
// MyMethod1
C_OBJECT($0)
$0:=New object("a";10;"b";20)

//myMethod2
$result:=MyMethod1.a //10
```

- コレクション 例:

```
myColl.length // コレクションの長さ
```

ポインター

注: オブジェクトは常に参照として渡されるため、通常はポインターを使用する必要はありません。オブジェクトを引数として渡す際、4D 内部では自動的にポインターに類似したメカニズムを使うことでメモリの消費を最小限に抑え、引数を編集して返すことを可能にします。つまり、ポインターは必要ないということです。それでもポインターを使用したい場合には、プロパティ値はポインターを通してアクセスすることができます。

ポインターを使ってオブジェクトプロパティにアクセスするには、直接オブジェクトを使用する場合と方法が似ていますが、"ドット" 記号は省略する必要があります。

- オブジェクト記法によるアクセス:

```
pointerOnObject->propertyName
```

- 大カッコを使用する方法:

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

例:

```
C_OBJECT(vObj)
C_POINTER(vPtr)
vObj:=New object
vObj.a:=10
vPtr:=->vObj
x:=vPtr->a //x=10
```

Null 値

オブジェクト記法を使用する場合、null 値は Null コマンドを通してサポートされています。このコマンドを使用すると、null 値をオブジェクトプロパティやコレクション要素に割り当てたり、それらと比較したりすることができます。例:

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

詳細については、 Null コマンドの説明を参照してください。

未定義の値

オブジェクトプロパティを評価した結果、未定義の値が生成されることがあります。未定義の式を読み込んだ、または割り当てようとしたときに 4D は通常、エラーを生成します。ただし以下の場合には生成されません:

- 未定義のオブジェクトやプロパティ値を読み込むと未定義 (undefined) が返されます。未定義の値を (配列を除く) 変数に割り当てるには、CLEAR VARIABLE コマンドを使うのと同じ効果があります:

```
C_OBJECT($o)
C_LONGINT($val)
$val:=10 // $val=10
$val:=$o.a // $o.a は未定義 (エラーなし) なため、この値を代入すると変数が初期化されます
// $val=0
```

- 未定義のコレクションの length プロパティは 0 を返します:

```
C_COLLECTION($c) // 変数は作成されたが、コレクションは未定義      $size:=$c.length // $size = 0
```

- 未定義の値を引数としてプロジェクトメソッドに渡した場合、宣言された引数の型に応じて、0 あるいは "" (空の文字列) へと自動変換されます。

```
C_OBJECT($o)
mymethod($o.a) // 未定義の引数を渡すと

// mymethod メソッド内では
C_TEXT($1) // 引数の型はテキスト
// $1 の中身は""
```

- 条件式で、If あるいは Case of キーワードで未定義と評価された場合には、自動的にfalse へと変換されます：

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

- 未定義の値を既存のオブジェクトプロパティに代入した場合、その値は型に応じて初期化、あるいは消去されます：
- オブジェクト、コレクション、ポインター： Null
- ピクチャー： 空のピクチャー
- ブール： False
- 文字列： ""
- 数値： 0
- 日付："オブジェクトではISO日付フォーマットの代わりに日付型を使用する" 設定が有効化されている場合は !00-00-00!、それ以外の場合には ""
- 時間： 0 (ミリ秒単位)
- 未定義、Null： 变化なし

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

- 未定義の値を存在しないオブジェクトのプロパティへと代入した場合は、何も起りません。

4Dコード内の式に対して特定の型であることが要求される場合、その式を適切な 4Dキャストコマンド (String , Num , Date , Time , Bool) で囲うことで、たとえ未定義に評価されたとしても正しい型を確実に得ることができます。これらのコマンドは式が未定義と評価された場合に、指定された型の空の値を返します。たとえば：

```
$myString:=Lowercase(String($o.a.b)) // 未定義の場合でもコード内でエラーが起きないように
// 文字列の値が得られるようにします
```

例題

オブジェクト記法を使用すると、オブジェクトを扱う際の 4Dコードを単純化することができます。同時に、コマンドベースの記法も引き続き完全にサポートされています。

- オブジェクトの読み書き (この例題ではオブジェクト記法とコマンド記法を比較します)：

```

// オブジェクト記法を使用
C_OBJECT($myObj) // 4Dオブジェクト変数を宣言
$myObj:=New object // オブジェクト作成し、変数に代入
$myObj.age:=56
$age:=$myObj.age // 56

// コマンド記法を使用
C_OBJECT($myObj2) // 4Dオブジェクト変数を宣言
OB SET($myObj2;"age";42) // オブジェクトを作成し、ageプロパティを追加
$age:=OB Get($myObj2;"age") // 42

// もちろん両方の記法を混用することもできます
C_OBJECT($myObj3)
OB SET($myObj3;"age";10)
$age:=$myObj3.age // 10

```

- プロパティの作成と、オブジェクトを含む値の代入:

```

C_OBJECT($Emp)
$Emp:=New object
$Emp.city:="London" // cityプロパティを作成し、その値を"London"に設定します
$Emp.city:="Paris" // cityプロパティを変更します
$Emp.phone:=New object("office";"123456789";"home";"0011223344")
// phoneプロパティを作成し、その値にオブジェクトを設定します

```

- オブジェクト記法を使用すると、サブオブジェクトの値を簡単に取得できます:

```

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

```

- 大カッコ [] を使用すると文字列を使ってプロパティにアクセスできます:

```

$Emp["city"]:="Berlin" // city プロパティを変更
// これは変数を通してプロパティを作成する場合に便利です
C_TEXT($addr)
$addr:="address"
For($i;1;4)
    $Emp[$addr+String($i)]:=""
End for
// $Emp object には4つの空のプロパティ "address1...address4" が作成されました

```

ピクチャー

ピクチャーのフィールド・変数・式に格納されるデータは、任意の Windows または Macintosh の画像です。これらの画像には、ペーストボード上に置いたり、4Dコマンドやプラグインコマンド（`READ PICTURE FILE` など）を使用してディスクから読み出すことのできる画像を含みます。

4D は Windows と macOS の両方においてネイティブな API を使用してフィールドや変数のピクチャーをエンコード（書き込み）およびデコード（読み込み）します。これらの実装は現在デジタルカメラで使用されている RAW フォーマット含め、数多くのネイティブなフォーマットへのアクセスを提供します。

- Windows では、4DはWIC (Windows Imaging Component) を使用します。
- macOS では、4D は ImageIO を使用します。

WIC および ImageIO はピクチャー内のメタデータの書き込みを許可しています。`SET PICTURE METADATA` および `GET PICTURE METADATA` コマンドを使用することで、それらのメタデータを開発に役立てることができます。

ピクチャー Codec ID

4D は多様な [ピクチャーフォーマット](#) をネイティブにサポートします: .jpeg, .png, .svg 等。

4D が認識するピクチャーフォーマットは `PICTURE CODEC LIST` コマンドからピクチャー Codec IDとして返されます。これは以下の形式で返されます:

- 拡張子 (例: ".gif")
- MIME タイプ (例: "image/jpeg")

それぞれのピクチャーフォーマットに対して返される形式は、当該 Codec が OS レベルで記録されている方法に基づきます。エンコーディング（書き込み）用コーデックにはライセンスが必要な場合があるため、利用できるコーデックの一覧は、読み込み用と書き込み用で異なる可能性があることに注意してください。

多くの [4D ピクチャー管理コマンド](#) は Codec ID を引数として受けとることができます。したがって、`PICTURE CODEC LIST` から返されるシステムID を使用しなければなりません。4D が認識するピクチャーフォーマットは `PICTURE CODEC LIST` コマンドによって返されます。

ピクチャー演算子

演算子	シンタックス	戻り値	動作
水平連結	Pict1 + Pict2	ピクチャー	Pict1 の右側に Pict2 を追加します
垂直連結	Pict1 / Pict2	ピクチャー	Pict1 の下側に Pict2 を追加します
排他的論理和	Pict1 & Pict2	ピクチャー	Pict1 の前面に Pict2 を重ねます (Pict2 が前面) COMBINE PICTURES(pict3;pict1;Superimposition;pict2) と同じ結果になります。
包括的論理和	Pict1 Pict2	ピクチャー	Pict1 と Pict2 を重ね、そのマスクした結果を返します (両ピクチャーとも同じサイズである必要があります) \$equal:=Equal pictures(Pict1;Pict2;Pict3) と同じ結果になります。
水平移動	ピクチャー + 数値	ピクチャー	指定ピクセル分、ピクチャーを横に移動します。
垂直移動	ピクチャー / 数値	ピクチャー	指定ピクセル分、ピクチャーを縦に移動します。
サイズ変更	ピクチャー * 数値	ピクチャー	割合によってピクチャーをサイズ変更します。
水平スケール	ピクチャー *+ 数値	ピクチャー	割合によってピクチャー幅をサイズ変更します。
垂直スケール	ピクチャー * 数値	ピクチャー	割合によってピクチャー高さをサイズ変更します。

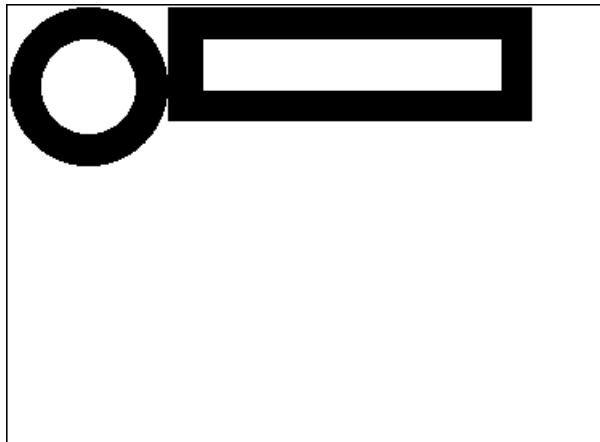
注:

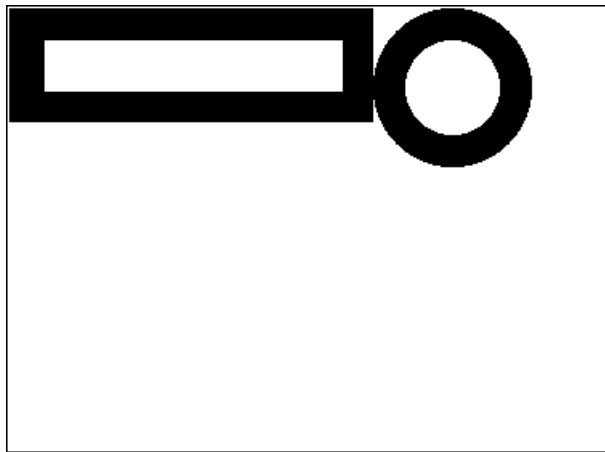
- | 演算子を使用するためには、Pict1 と Pict2 が完全に同一のサイズでなければなりません。二つのピクチャーサイズに違いがある場合、Pict1 | Pict2 は空のピクチャーを生成します。
- COMBINE PICTURES コマンドは、それぞれのソースピクチャーの特性を結果ピクチャーに保持しつつ、ピクチャーの重ね合わせをおこないます。
- TRANSFORM PICTURE コマンドを使って、さらなる画像処理をおこなうことができます。
- ピクチャー用の比較演算子はありませんが、Equal picture コマンドを使って 2つのピクチャーを比較することができます。

例題

水平連結

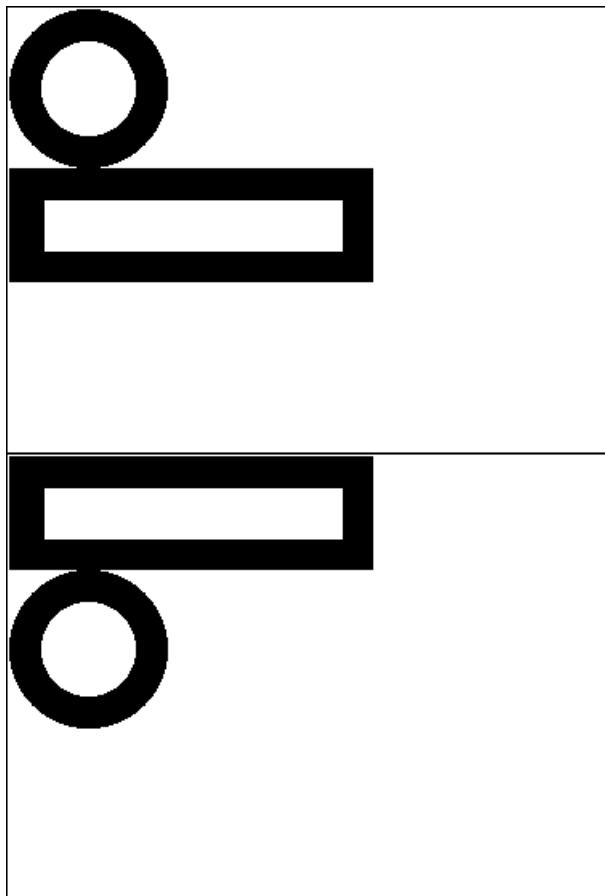
```
circle+rectangle // circle の右に rectangle が追加されます。
rectangle+circle // rectangle の右に circle が追加されます。
```





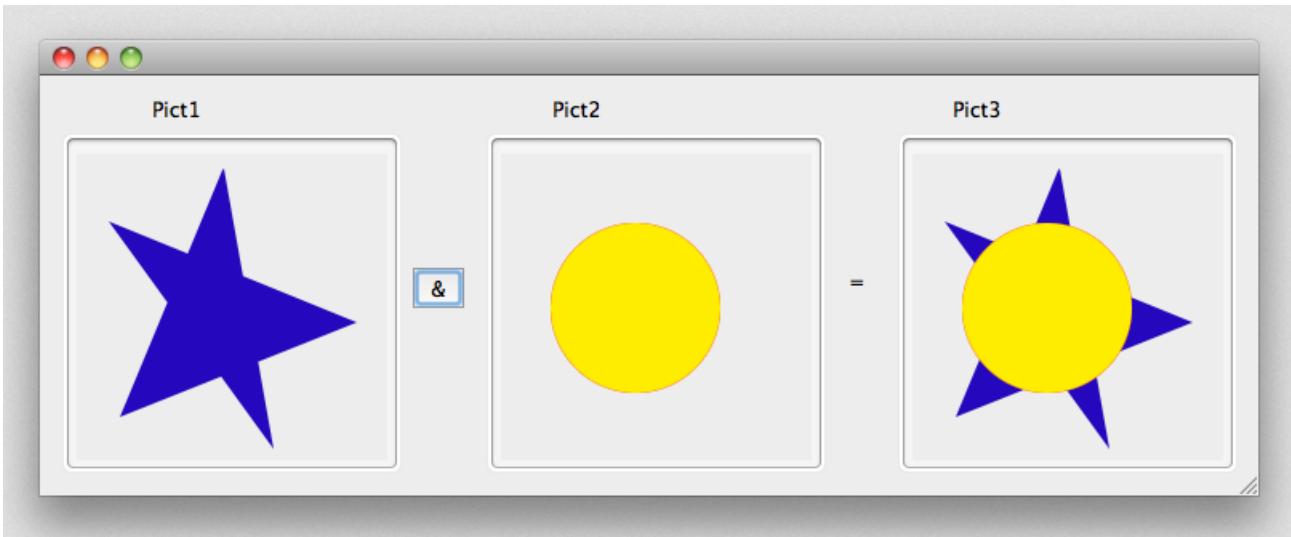
垂直連結

```
circle/rectangle // circle の下に rectangle が追加されます。  
rectangle/circle // rectangle の下に circle が追加されます。
```



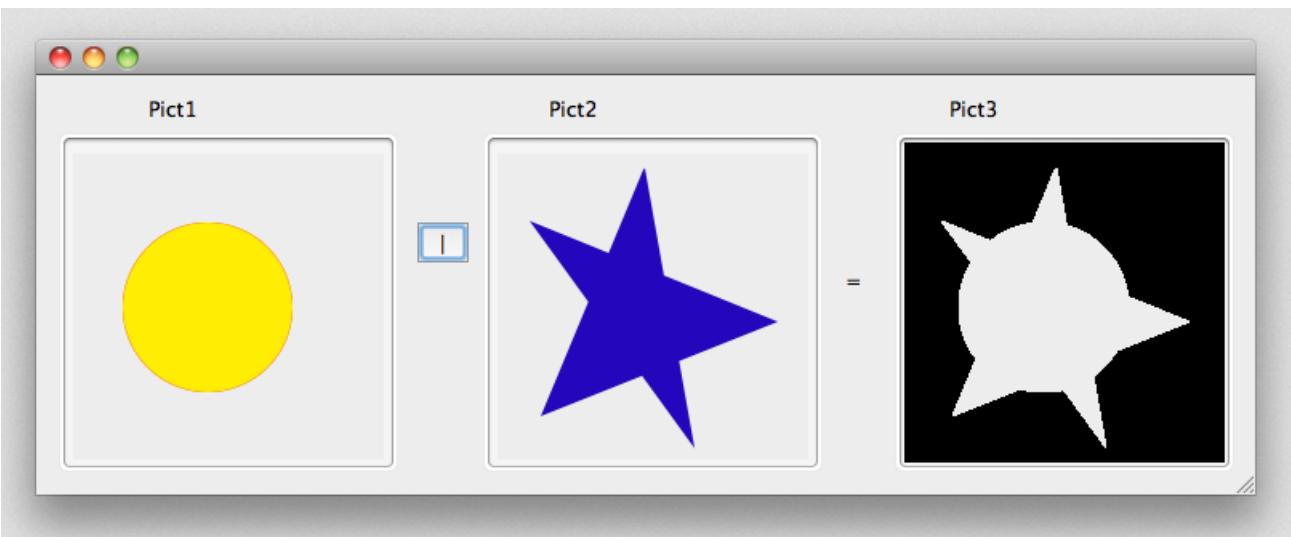
排他的論理和

```
Pict3:=Pict1 & Pict2 // Pict1 の上に Pict2 を重ねます。
```



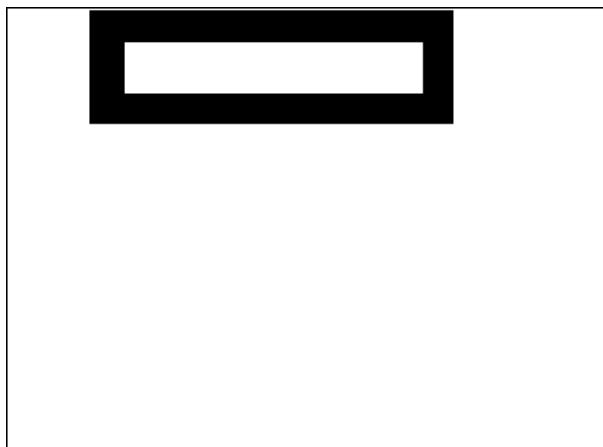
包括的論理和

```
Pict3:=Pict1|Pict2 // 同じサイズの二つのピクチャーを重ね合わせた上でそのマスクの結果を返します。
```



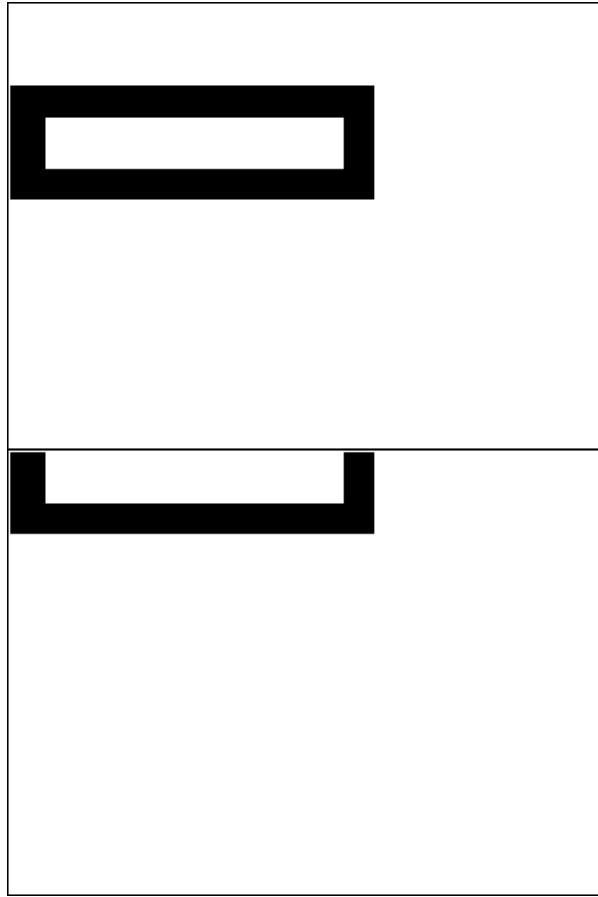
水平移動

```
rectangle+50 // rectangle を右に 50ピクセル移動します。  
rectangle-50 // rectangle を左に 50ピクセル移動します。
```



垂直移動

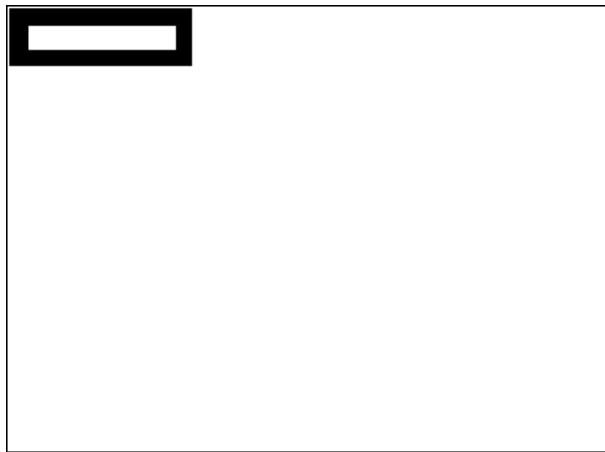
```
rectangle/50 // rectangle を下に 50ピクセル移動します。  
rectangle/-20 // rectangle を上に 20ピクセル移動します。
```



拡大

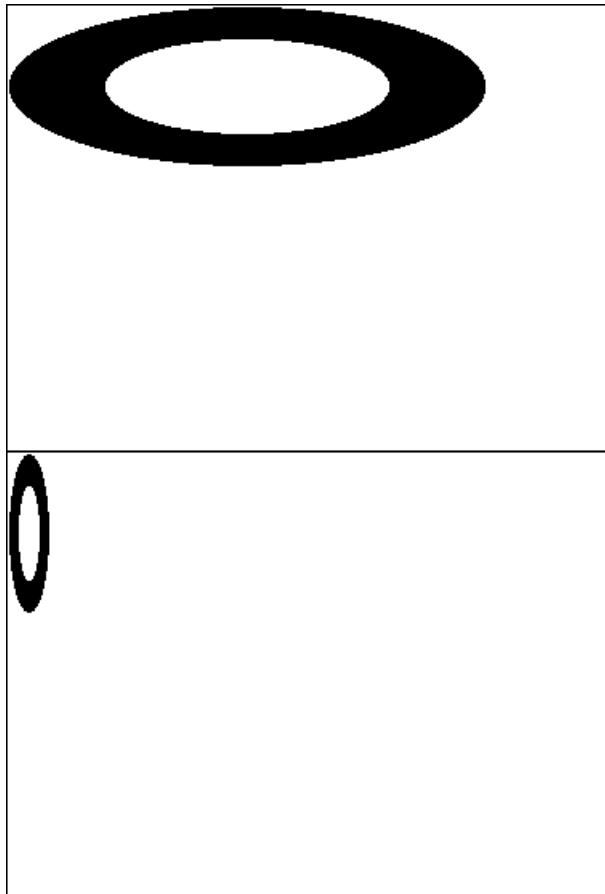
```
rectangle*1.5 // rectangle を 50%拡大します。  
rectangle*0.5 // rectangle を 50%縮小します。
```





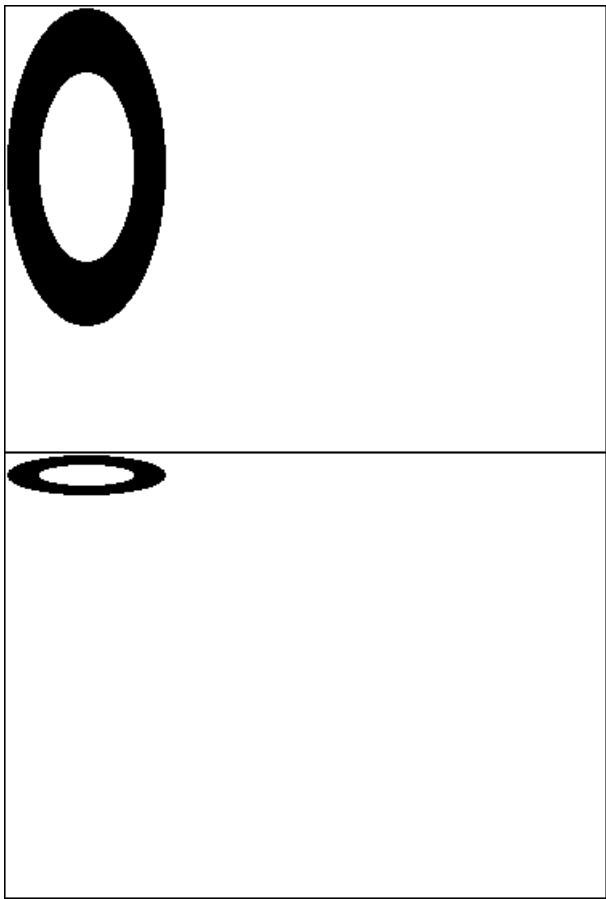
水平スケール

```
circle*+3 // circle の幅を 3倍に広げます。  
circle*+0.25 // circle の幅を 25%に縮めます。
```



垂直スケール

```
circle*|2 // circle の高さを 2倍に伸ばします。  
circle*|0.25 // circle の高さを 25%に縮めます。
```



ポインター

ポインターの変数や式は、別の変数（配列、配列要素を含む）、テーブル、またはフィールドへの参照です。ポインタータイプのフィールドは、存在しません。

ポインターは、（プログラミングにおける）データを参照するための高度な方法を提供します。4D ランゲージ使用時にテーブル・フィールド・変数・配列等にアクセスするには、単純に名前を用います。ですが、名前を使用しないでデータを参照する、またはアクセスした方が便利な場合もあります。ポインターを使うとこれが実現できます。

ポインターの背景にある概念は、日常生活でもよく使われています。対象物を正確に知らないまま、それを示すことがあります。たとえば、友人に対して "登録番号123ABDの車に乗ろう" と言わずに "君の車に乗ろう" と言う場合です。つまり、"登録番号123ABDの車" を "君の車" で示したわけです。この場合、"登録番号123ABDの車" はオブジェクトの名前で、"君の車" はオブジェクトを参照するためのポインターと考えることができます。

対象物を明示しないで参照できると、非常に便利です。たとえば、友人が新しい車に買い替えても、同じく "君の車" と言うことができます。ポインターも同じように機能します。たとえば、同じポインターがある時は数値フィールド "Age" を参照し、別の時には数値変数 "Old Age" を参照することもできます。いずれの場合にもポインターは数値データを参照しており、それは計算に使用することができます。

テーブル・フィールド・変数・配列・配列要素・オブジェクトを参照するためにポインターを使用することができます。以下の表に、各タイプの例を示します：

タイプ	参照時	使用時	代入時
テーブル	vpTable:=->[Table]	DEFAULT TABLE(vpTable->)	n/a
フィールド	vpField:=->[Table]Field	ALERT(vpField->)	vpField->:="John"
変数	vpVar:=->Variable	ALERT(vpVar->)	vpVar->:="John"
配列	vpArr:=->Array	SORT ARRAY(vpArr-> ; >)	COPY ARRAY (Arr;vpArr->)
配列要素	vpElem:=->Array{1}	ALERT (vpElem->)	vpElem->:="John"
オブジェクト	vpObj:=->myObject	ALERT (vpObj->myProp)	vpObj->myProp:="John"

ポインターの基本

ポインターの使用方法について例題を用いて説明します。以下の例は、ポインターを通して変数にアクセスする方法を示します。まず、変数を作成します：

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

\$MyVar は、文字列 "Hello" を含む変数です。\$MyVar に対するポインターを作成します：

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

ポインター記号 (->) は、"…に対するポインターを求める" ことを意味します。この記号は、"ダッシュ" (-) の後に "大なり" (>) を付けて構成されます。ここでは、\$MyVar を参照するポインターを取得します。このポインターは、代入演算子 (:=) で \$MyPointer に対して割り当てられます。

\$MyPointer は、\$MyVar に対するポインターを格納する変数です。\$MyPointer は、"Hello" という \$MyVar の値を含みませんが、その値を参照することはできます。以下の式は \$MyVar の値を返します：

```
$MyPointer->
```

前述の式は、"Hello" という文字列を返します。ポインター記号 (->) をポインターの後につけると、参照先の値を取得することができます。これをデリファレンス（参照外し）と呼びます。

ポインター記号 (->) を後につけたポインターは、その参照先を直接使うのと同義であることを理解することが重要です。つまり、変数 \$MyVar を使用することと、\$MyPointer-> を使用することは、まったく同じ意味になります。たとえば、以下のステートメントはアラートボックスに文字列 "Hello" を表

示します:

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

\$MyPointer を使用して \$MyVar の値を変更することもできます。下記のステートメントは、変数 \$MyVar に文字列 "Goodbye" を代入します:

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

この2つの \$MyPointer-> を使用した例のとおり、\$MyVar を使用するのとまったく同じ動作が実行されます。以下の2つのステートメントも、同一の動作を実行します。両方とも、変数 \$MyVar の現在の値をアラートボックスに表示します:

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

以下の2つのステートメントも、同一の動作を実行します。両方とも \$MyVar に、文字列 "Goodbye" を代入します:

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

ポインター演算子

前提:

```
// vPtrA と vPtrB は同じ対象を参照します
vPtrA:=->anObject
vPtrB:=->anObject
// vPtrC は別の対象を参照します
vPtrC:=->anotherObject
```

演算子	シンタックス	戻り値	式	値
等しい	ポインター = ポインター	ブール	vPtrA = vPtrB	True
			vPtrA = vPtrC	False
異なる	ポインター # ポインター	ブール	vPtrA # vPtrC	True
			vPtrA # vPtrB	False

ポインターの使用例

テーブルへのポインター

テーブルの代わりにデリファレンスしたポインターを使用することができます。以下のようなステートメントで、テーブルのポインターを作成します:

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

あるいは、以下のように `Table` コマンドを使用してテーブルのポインターを得ることができます:

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

取得したポインターは、以下のようにデリファレンスしてコマンドに渡すことができます:

```
DEFAULT_TABLE($TablePtr->)
```

フィールドへのポインター

フィールドの代わりにデリファレンスしたポインターを使用することができます。以下のようなステートメントで、フィールドのポインターを作成します：

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

あるいは、以下のように `Field` コマンドを使用してフィールドのポインターを得ることができます：

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

取得したポインターは、以下のようにデリファレンスしてコマンドに渡すことができます：

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

変数へのポインター

プロセス変数またはローカル変数のポインターを使う場合、参照される変数はポインターが使用される時点ですでに定義されていなければなりません。ローカル変数は、それを作成したメソッドの実行が終わると破棄され、プロセス変数もそれを作成したプロセスの終了時に削除される点に留意してください。存在しない変数をポインターで呼び出そうとすると、インタープリターモードでは（「変数が設定されていません」という内容の）シンタックスエラーが起きます。コンパイルモードでは、さらに重大なエラーが発生する可能性があります。

ローカル変数のポインターを使用すると、プロセス変数の使用を控えることができます。ローカル変数へのポインターは、同じプロセス内でのみ使用することができます。デバッガーにおいて、別のメソッドで宣言されたローカル変数へのポインターを表示すると、ポインターの後ろの括弧内にそのメソッド名が表示されます。例として、Method1 で以下のように書いたとします：

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

Method2 実行中のデバッガーは \$1 を次のように表示します：

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

\$1 の値は、次のようになります：

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

配列要素へのポインター

配列要素に対するポインターを作成することができます。以下の例は配列を作成し、配列の最初の要素を指し示すポインターを変数 `$ElemPtr` に割り当てます：

```
ARRAY REAL($anArray;10) // 配列を作成  
$ElemPtr:=->$anArray{1} // 配列要素へのポインターを作成
```

以下のように、ポインターの参照先である配列要素に値を代入することができます：

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

配列へのポインター

配列に対するポインターを作成することができます。以下の例は配列を作成し、配列を指し示すポインターを変数 \$ArrPtr に割り当てます：

```
ARRAY REAL($anArray;10) // 配列を作成  
$ArrPtr:==>$anArray // 配列へのポインターを作成
```

ポインターの参照先はあくまでも配列であり、配列要素ではないことを理解することが重要です。たとえば、デリファレンスしたポインターを以下のように使用できます：

```
SORT ARRAY($ArrPtr->;>) // 配列の並べ替え
```

配列の4番目の要素にアクセスするのに配列のポインターを使う場合は、以下のように記述します：

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

メソッドの引数としてのポインター

ポインターは引数としてメソッドに渡すことができます。メソッド内で、ポインターの参照先の値を変更することができます。たとえば、以下のメソッド `takeTwo` は、2つのポインターを引数として受け取ります。そして、最初の引数の参照先を大文字に変換し、2つめの引数の参照先を小文字に変換します。当該プロジェクトメソッドのコードです：

```
//takeTwo プロジェクトメソッド  
//$1 - 文字列フィールドまたは変数へのポインター。 これを大文字に変換します。  
//$2 - 文字列フィールドまたは変数へのポインター。 これを小文字に変換します。  
$1->:=Uppercase($1->)  
$2->:=Lowercase($2->)
```

以下のステートメントではメソッド `takeTwo` を使用し、フィールドの値を大文字に、変数の値を小文字に変換します：

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

このフィールド [myTable]myField の値が "jones" であれば、"JONES" に変更されます。他方、変数 \$MyVar の値が "HELLO" であれば、"hello" に変更されます。

メソッド `takeTwo` で宣言されている引数の型と、引数として渡したポインターの参照先のデータタイプが一致していることが重要です。この例では、ポインターの参照先は必ず文字列またはテキスト型でなければなりません。

ポインターへのポインター

より複雑な使い方として、ポインターを参照するポインターを使うことができます。以下の例を考えます：

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

この例はアラートボックスに "Goodbye" を表示します。

各行について見てきましょう：

- `$MyVar:="Hello" -->` この行は、変数 \$MyVar に "Hello" という文字列を代入しています。
- `$PointerOne:==>$MyVar -->` 変数 \$PointerOne に、変数 \$MyVar へのポインターを代入します。
- `$PointerTwo:==>$PointerOne -->` 新たな変数 \$PointerTwo に、\$MyVar を参照する \$PointerOne へのポインターを代入します。

- `($PointerTwo->)->:="Goodbye"` --> `$PointerTwo->` は `$PointerOne` を示し、`$PointerOne` は `$MyVar`を示します。つまり、`($PointerTwo->)->` は、`$MyVar` を示しています。結果として、文字列 "Goodbye" が `$MyVar` に代入されます。
- `ALERT (($PointerTwo->)->)` --> 先の説明と同様に `$PointerTwo->` は `$PointerOne` を示し、`$PointerOne` は `$MyVar`を示しています。つまり、`($PointerTwo->)->` は、`$MyVar` を示しています。結果としてアラートボックスには `$MyVar` の内容が表示されます。

以下の例では、`$MyVar` に "Hello" が代入されます:

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

以下のステートメントは、`$NewVar` に `$MyVar` の値である "Hello" が代入されます:

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

重要: デリファレンスを複数おこなうには括弧が必要です。

文字列

文字列とは、以下を示す総称です：

- テキストフィールドまたは変数：テキストフィールド、変数、または式には 0～2 GB のテキストを格納することができます。
- 文字フィールド：文字フィールドには 0～255 文字までの文字を格納することができます（上限はフィールドが定義されたときに設定されます）。

文字列リテラル

文字列リテラル定数は、次のように二重引用符 ("...") で囲んで表します。文字列定数の例を次に示します：

```
"レコード追加"  
"レコードが見つかりません"  
"送り状"
```

空の文字列は、2つの引用符の間に何も入れない状態 ("") で表します。

エスケープシーケンス

以下のエスケープシーケンスを文字列内で使用できます：

エスケープシーケンス	意味する文字
¥n	LF (行送り)
¥t	HT (タブ)
¥r	CR (改行)
¥¥	¥ (バックスラッシュ)
¥"	" (引用符)

注：¥ (バックスラッシュ) は Windows でパス名の区切り文字として使用されています。通常 4D はメソッドエディターに入力されたバックスラッシュを自動で "¥" に置き換えることで、これを正しく解釈します。例えば "C:¥Folder" と入力すると "C:¥¥Folder" に変換されます。しかし "C:¥MyDocuments¥New" と入力した場合、4Dは二番目のバックスラッシュは "¥N" (行送り) と解釈してしまい、"C:¥¥MyDocuments¥New" を表示します。このようなケースでは開発者がバックスラッシュを2つ入力するようにならなければなりません。さらに正規表現のパターン定義でもバックスラッシュがエスケープシーケンスとして使用されます。正規表現パターン "¥" を4Dのメソッドエディターに記述する場合は "¥¥" となる点に注意してください。

文字列演算子

演算子	シンタックス	戻り値	式	値
連結 (結合)	文字列 + 文字列	String	"abc" + "def"	"abcdef"
繰り返し	文字列 * 数値	String	"ab" * 3	"ababab"
等しい	文字列 = 文字列	ブール	"abc" = "abc"	True
			"abc" = "abd"	False
異なる	文字列 # 文字列	ブール	"abc" # "abd"	True
			"abc" # "abc"	False
大きい	文字列 > 文字列	ブール	"abd" > "abc"	True
			"abc" > "abc"	False
小さい	文字列 < 文字列	ブール	"abc" < "abd"	True
			"abc" < "abc"	False
以上	文字列 >= 文字列	ブール	"abd" >= "abc"	True
			"abc" >= "abd"	False
以下	文字列 <= 文字列	ブール	"abc" <= "abd"	True
			"abd" <= "abc"	False
キーワードを含む	文字列 % 文字列	ブール	"Alpha Bravo" % "Bravo"	True
			"Alpha Bravo" % "ravo"	False
	ピクチャー % 文字列	ブール	Picture_expr % "Mer"	True (*)

(*) キーワード "Mer" がピクチャー式 (フィールドまたは変数) に格納されたピクチャーの IPTC/Keywords メタデータに含まれている場合。

文字列比較の詳細

- 文字列は文字ごとに比較されます (後述の [キーワード](#) による検索の場合を除きます)。
- 文字列が比較されるとき文字の大小文字は無視されます。したがって、"a"="A"は `true` を返します。大文字と小文字を区別して比較するには、文字コードで比較してください。例えば次の式は `FALSE` です:

```
Character code("A")=Character code("a")
// Character code("A") は 65
// Character code("a") は 97
```

- 文字列が比較される場合、アクセント等の発音区別符号は無視されます。たとえば、日本語においては以下の式は `true` を返します:

```
"あ"="ア"
"ア"="ア"
"�"="1"
```

注: 文字列比較にあたっては、4D のデータファイルに定義された 言語の特性が考慮されます。これは、システムの使用言語とは異なる場合があります。

ワイルドカード記号 (@)

4D ランゲージでは @ をワイルドカード記号として使用します。ワイルドカードは、すべての文字列の比較に使用することができ、ワイルドカードによって置き換わる文字の数は指定されません。たとえば、次の式は `true` になります:

```
"abcdefghijklm"="abc@"
```

ただし、複数の文字を比較する目的のワイルドカード記号は、比較演算子の右側の式で使用しなければなりません。比較演算子の左側の式においては、"@" は単なる文字であると解釈されます。たとえば、次の式は `FALSE` です:

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

ワイルドカードは “0文字以上” を意味します。以下の式はすべて `true` です:

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

一方、どのような場合でも、ワイルドカードを 2つ連続して使用した文字列比較は常に `FALSE` を返します。次の式は `FALSE` になります:

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

比較演算子が < あるいは > 記号である、あるいはこれらを含む場合、演算子の右側の式の終りに置かれた1つのワイルドカードのみサポートされています:

```
"abcd"<="abc@"
"abcd"<="abc@ef" // 有効な比較ではありません
```

文字列の比較または検索において、@ をワイルドカードではなく一般の文字として扱いたい場合、`Character code (At sign)` 指示を使用します。たとえば、文字列が @ 文字で終わっているかどうかを知りたいとします。以下の式は (`$vsValue` が空でなければ) 常に `true` です:

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

以下のようにすると、式は意図したように評価されます:

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

注: ストラクチャー設定のデータベースページには、文字列に @ 記号が含まれているとき、それをどう解釈するかを指定するオプションが提供されています。

キーワード

他の文字列比較と異なり、"%" 記号を使ったキーワードによる検索はテキスト中の単語を検索します: 単語は一つのまとまりとして個々に扱われます。複数の単語や、音節など単語の一部を検索するような場合、% 演算子は常に `false` を返します。区切り文字 (スペースや句読点など) に囲まれた文字列が単語として認識されます。“Today's” のようにアポストロフィを含む単語は、通常それを含めた 1つの単語として扱われますが、特定の場合には無視されます (以下の注記を参照ください)。数字も検索できます。小数点は区切り文字ではなく、数字の一部として扱われます。ただし、通貨や温度などを表す記号は無視されます。

```
"Alpha Bravo Charlie%"Bravo" // true
"Alpha Bravo Charlie%"vo" // false
"Alpha Bravo Charlie%"Alpha Bravo" // false
"Alpha,Bravo,Charlie%"Alpha" // true
"Software and Computers%"comput@" // true
```

注:

- 4Dは、<>=# 演算子を使った文字列比較や、キーワードの検出にICUライブラリを使用しています。実装されているルールの詳細に関しては、以下のアドレスを参照して下さい: http://www.unicode.org/unicode/reports/tr29/#Word_Boundaries

- 日本語版の 4Dでは、ICU の代わりにデフォルトで Mecab が使用されています。詳細な情報に関しては、[Mecab のサポート\(日本語版\)](#) を参照してください。

文字参照記号

文字参照記号: [[...]]

文字参照記号は、文字列から一文字のみを参照するのに使用します。この構文は、テキストおよび文字列型の変数やフィールドの任意の位置の文字を指し示します。

文字参照記号が代入演算子 (:=) の左側にある場合、文字列の指定した位置に一文字を代入します。以下の例は、vsName が空の文字列ではない場合に、vsName の最初の文字を大文字にします。

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

それ以外の場合には、式内で使用される文字列参照記号は、参照する文字を1文字の独立した文字列として返します。たとえば:

```
// 以下の例は vtText の最後の文字が "@" であるかをテストします。
If(vtText#"")
  If(Character code(Substring(vtText;Length(vtText);1))=At sign)
  // ...
  End if
End if

// 文字参照記号を使用し、よりシンプルに記述できます:
If(vtText#"")
  If(Character code(vtText[[Length(vtText)]])=At sign)
  // ...
  End if
End if
```

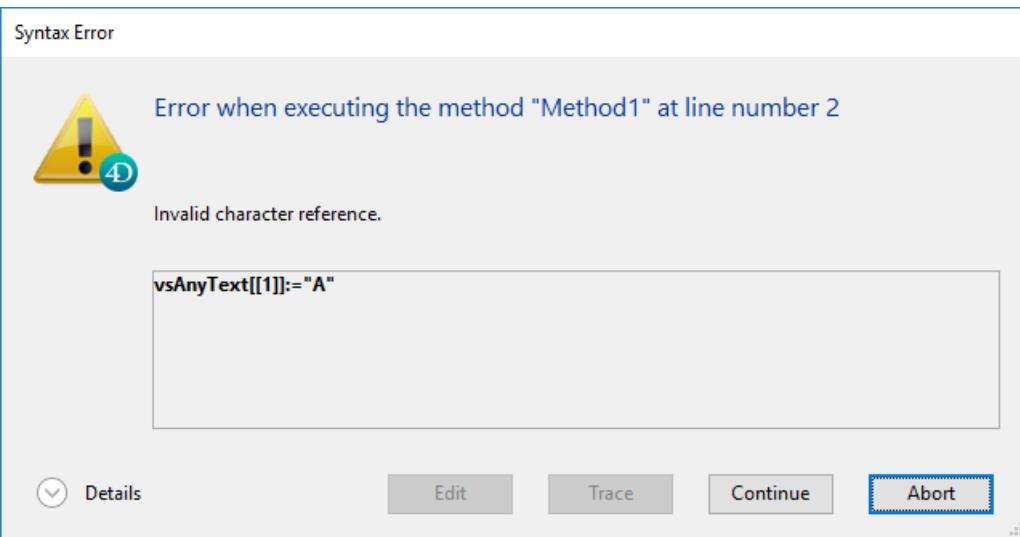
無効な文字列参照に関する注意

文字列参照記号を使用する際、配列に存在する要素を指定するのと同じ要領で、文字列内に存在する文字を指定しなければなりません。たとえば、文字列変数の20番目の文字を参照する場合、この変数は必ず、少なくとも20文字以上の長さがなくてはなりません。長さが足りない場合:

- インターブリターモードでは構文エラーは発生しません。
- コンパイルモードで範囲チェックオプションを無効にしている場合には、メモリ領域を破壊するおそれがあります。
- コンパイルモードで範囲チェックオプションを有効にしている場合には、エラーが発生します。たとえば、次のように文字列やテキストの終点を超えた位置に文字を書き込むコードを実行すると:

```
// 真似をしないでください
vsAnyText:=""
vsAnyText[[1]]:="A"
```

ランタイムにおいてエラーがトリガーされます:



例題

以下のプロジェクトメソッドは、文字列内の各単語の先頭文字を大文字に変換し、結果の文字列を返します。

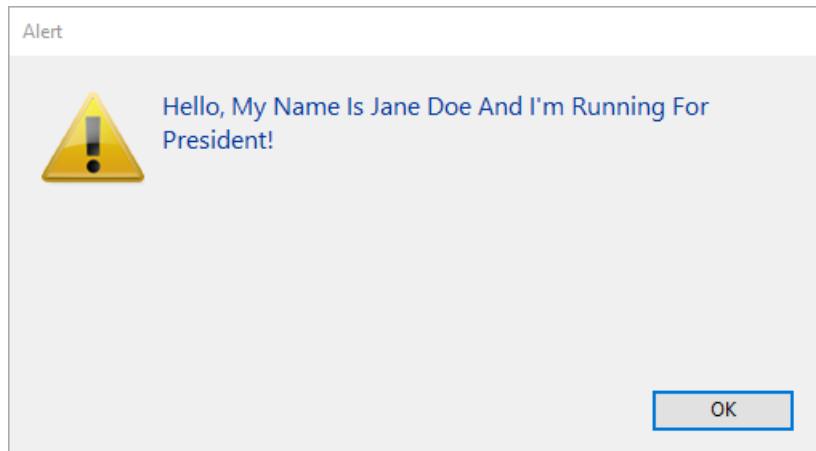
```
// Capitalize_text プロジェクトメソッド
// Capitalize_text ( Text ) -> Text
// Capitalize_text ( Source text ) -> Capitalized Text

$0:=$1
$vlLen:=Length($0)
If($vlLen>0)
    $0[[1]]:=Uppercase($0[[1]])
    For($vlChar;1;$vlLen-1)
        If(Position($0[[ $vlChar]]; " !&()-{}:;<>?/,.=+*")>0)
            $0[[ $vlChar+1]]:=Uppercase($0[[ $vlChar+1]])
        End if
    End for
End if
```

使用例:

```
ALERT(Capitalize_text("hello, my name is jane doe and i'm running for president!"))
```

以下のように表示されます:



時間

時間のフィールド・変数・式の範囲は 00:00:00 から 596,000:00:00 までです。

時間は 24時間制です。

時間の値は数値として扱うことができます。時間から返される数値は、その時間が表す総秒数です。

注: 4Dランゲージリファレンス マニュアルでは、コマンド説明における時間引数は特に明記されていない限り、「時間」と表記されています。

時間リテラル

時間リテラル定数は、疑問符 (?) ... (?) で囲んで表します。

時間は、“時:分:秒” の順で表し、それぞれをコロン (:) で区切ります。時間は24時間制で指定します。

時間定数の例を次に示します:

```
?00:00:00? // 午前0時  
?09:30:00? // 午前9時30分  
?13:01:59? // 午後1時1分59秒
```

空の時間は、?00.00.00? のように指定します。

Tip: メソッドエディターでは空の時間を入力するためのショートカットが提供されています。空の時間を入力するには、疑問符 (?) の入力後に Enter キーを押します。

時間演算子

演算子	シンタックス	戻り値	式	値
加算 (足し算)	時間 + 時間	時間	?02:03:04? + ?01:02:03?	?03:05:07?
減算 (引き算)	時間 - 時間	時間	?02:03:04? - ?01:02:03?	?01:01:01?
加算 (足し算)	時間 + 数値	数値	?02:03:04? + 65	7449
減算 (引き算)	時間 - 数値	数値	?02:03:04? - 65	7319
乗算 (かけ算)	時間 * 数値	数値	?02:03:04? * 2	14768
除算 (割り算)	時間 / 数値	数値	?02:03:04? / 2	3692
倍長整数を返す除算	時間 ¥ 数値	数値	?02:03:04? ¥ 2	3692
モジューロ	時間 % 時間	時間	?20:10:00? % ?04:20:00?	?02:50:00?
モジューロ	時間 % 数値	数値	?02:03:04? % 2	0
等しい	時間 = 時間	ブール	?01:02:03? = ?01:02:03?	True
			?01:02:03? = ?01:02:04?	False
異なる	時間 # 時間	ブール	?01:02:03? # ?01:02:04?	True
			?01:02:03? # ?01:02:03?	False
大きい	時間 > 時間	ブール	?01:02:04? > ?01:02:03?	True
			?01:02:03? > ?01:02:03?	False
小さい	時間 < 時間	ブール	?01:02:03? < ?01:02:04?	True
			?01:02:03? < ?01:02:03?	False
以上	時間 >= 時間	ブール	?01:02:03? >=?01:02:03?	True
			?01:02:03? >=?01:02:04?	False
以下	時間 <= 時間	ブール	?01:02:03? <=?01:02:03?	True
			?01:02:04? <=?01:02:03?	False

例題 1

時間式を数値と組み合わせた式から時間式を取得するには、`Time` コマンドと `Time string` コマンドを使用します。

`Time` または `Current time` コマンドを使用する際に、時間型と数値型の式を組み合わせることができます：

```
// 以下の行は $vlSeconds に、深夜0時から現在の
// 1時間後までに経過した秒数を代入します。
$vlSeconds:=Current time+3600
// 以下の行は $vhSoon に 1時間後の時刻を代入します。
$vhSoon:=Time(Current time+3600)
```

2番目の行はより簡単に記述することができます：

```
// 以下の行は $vhSoon に1時間後の時刻を代入します。
$vhSoon:=Current time+?01:00:00?
```

例題 2

モジューロ演算子を使用できます。とくに24時間フォーマットを考慮した時間の追加に便利です：

```
$t1:=?23:00:00? // これは 23:00です  
// 2時間30分を追加します  
$t2:=$t1 +?02:30:00? // 単純な追加を行うと $t2 は ?25:30:00? になります。  
$t2:=($t1 +?02:30:00?)%?24:00:00? // $t2 は ?01:30:00? となります (翌朝の 01:30)
```

バリアント

バリアント型は、サポートしている型の任意のデータを受け取ることができる変数型です。一般的には、返したり受け取ったりする値の型が未定である汎用的なコードを書くためにこの変数型が使用されます。これは、たとえばオブジェクトの属性を扱うようなコードがそれに該当します。

バリアント型の変数は以下のデータタイプの値を格納することができます：

- BLOB
- ブール
- コレクション
- 日付
- 倍長整数
- オブジェクト
- ピクチャー
- ポインター
- 実数
- テキスト
- 時間
- null
- 未定義

バリアント型変数に配列を格納することはできません。

インターペリターモード、コンパイルモードのどちらにおいても、一つのバリアント型変数に異なる型のコンテンツを代入することができます。固定されている型の変数とは異なり、バリアント型変数の中身は変数自身の型（つまり、バリアント型）と同一ではありません。たとえば：

```
C_VARIANT($variant)

$variant:="hello world"
$type:=Type($variant) // 12 (Is variant)
$typeVal:=Value type($variant) // 2 (Is text)

$variant:=42
$type:=Type($variant) // 12 (Is variant)
$typeVal:=Value type($variant) // 1 (Is real)
```

変数が期待されるところであれば、どこでもバリアント型変数を使用することができます。ただし、変数の中身のデータ型は予期される型でなくてはなりません。また、バリアント型変数を使う場合、その時点で変数に格納されている値のみが実質的に使われます。たとえば：

```
C_VARIANT($v)
$v:="hello world"
$v2:=$v // バリアント型変数を、型未指定の変数に代入します

$t:=Type($v) // 12 (Is variant) 代入元の変数はバリアント型ですが
$t2:=Type($v2) // 2 (Is text) 代入先の変数はテキスト型です
```

バリアント型は、様々なタイプになりうるメソッドの引数（\$0, \$1,...）を宣言するのに使用できます。この場合、引数の値の型の確認作業が必要になります：

```
C_VARIANT($1)
Case of
: (Value type($1)=Is longint)
...
: (Value type($1)=Is text)
...
End case
```

バリエント型変数が必要ではない場合（つまりデータタイプが分かっている場合）、型が固定された変数を使用することが推奨されます。固定型変数の方がパフォーマンスやコードの可読性も良く、予期せぬデータタイプを見過ごしてしまった場合を防ぐため、コンパイラーにも優しいといえます。

変数

4D のデータは、根本的に異なる 2つの方法で保持されます。 フィールド はディスクに永続的にデータを保存するのに対し、変数 はメモリ上に一時的にデータを格納します。

データベースを作成する際には、フィールドに名前とデータタイプを指定します。 同様に、変数にも名前と [データタイプ](#) を指定します。

いったん作成された変数は、アプリケーションで必要とされる場所に使用できます。たとえば、テキスト変数と同じタイプのフィールドに格納するには次のように書きます：

```
[MyTable]MyField:=MyText
```

変数はランゲージの要素です。画面上に表示されることのない、裏方に徹した変数を作成・利用することができます。もちろん、フォーム上に変数の値を表示することもできます（ポインターやBLOBを除く）。また、変数に値を入力したり、変数の値をレポートに印刷したりすることも可能です。このとき、入力可や入力不可の変数オブジェクトはフィールドオブジェクトと同様に振舞い、提供されるコントロールも類似しています。フォーム上のボタン、リストボックス、スクロールエリア、ピクチャーボタンなどのオブジェクトも変数を使って制御することができるほか、保存不要な計算結果を表示させることもできます。

変数の宣言

変数は宣言によって作成されます。4D ランゲージでは、変数の宣言方法は2つあります：

- `var` キーワードを使った宣言（推奨、とくにオブジェクトやクラスをコードで使用する場合）
- "コンパイラ" や "配列" テーマの 4D ランゲージコマンドを使った宣言（クラシックランゲージのみ）

注：この方法は推奨されませんが、単純に使用することによって変数を宣言することもできます。正式にそれらを定義することは必須ではありません。たとえば、今日の日付に30日足した値を格納した変数を宣言するには、次のように書くことができます：

```
MyDate:=Current date+30 // MyDateを作成します
// これは日付型の変数であると 4D は推測します
// 30日後の日付が代入されます
```

`var` キーワードによる宣言

オブジェクト変数をクラスに紐づけることができるため、`var` キーワードを使った変数宣言が推奨されます。このシンタックスはコードエディターの自動補完機能を強化します。

`var` キーワードを使って変数を宣言するには、次のシンタックスを用います：

```
var <varName>{; <varName2>;...}{ : <varType>}
```

たとえば：

```
var $myText : Text // テキスト変数
var myDate1; myDate2 : Date // 複数の日付変数
var $myFile : 4D.File // File クラスオブジェクト変数
var $myVar // バリアント型変数
```

`varName` に指定する変数名は 4D の [識別子の命名規則](#) に従う必要があります。

このシンタックスは [ローカル変数とプロセス変数](#) の宣言のみサポートしています。[インタープロセス変数](#) および [配列](#) には使用できません。

`varType` には次が指定できます：

- [基本のデータ型](#)：変数には、宣言された型の値が格納されます
- [クラス参照](#) (4Dクラスまたはユーザークラス)：変数には、定義されたクラスのオブジェクトへの参照が格納されます

`varType` を省略すると、variant 型の変数が作成されます。

サポートされている varType 値の一覧です:

varType	内容
Text	テキスト値
Date	日付値
Time	時間値
Boolean	ブール値
Integer	倍長整数値
Real	実数値
Pointer	ポインター値
Picture	ピクチャーバリュー
BLOB	スカラーBLOB値
Collection	コレクション値
Variant	バリアント値
Object	デフォルトクラス (4D.Object) のオブジェクト
4D.<className>	4Dクラス名のオブジェクト
cs.<className>	ユーザークラス名のオブジェクト

例題

- 基本のデータ型の、ローカル変数およびプロセス変数の宣言:

```
var $myText; myText; $vt : Text
var myVar //variant

var $o : Object
// 次と同義です:
var $o : 4D.Object
// C_OBJECT($o) とも同義です
```

- 4Dクラス型のオブジェクト変数の宣言:

```
var $myFolder : 4D.Folder
var $myFile : 4D.File
```

- ユーザークラス型のオブジェクト変数の宣言:

```
var $myClass : cs.MyClass
var $dataclass : cs.Employee
var $entity : cs.EmployeeEntity
```

C_ 指示子による宣言

互換性に関する注記: メソッド内で変数を宣言するにあたって、この方法は推奨されません。var キーワードの使用が推奨されます。

"コンパイラ" テーマコマンドの指示子を使って、基本のデータ型の変数を宣言することができます。

たとえば、テキスト変数を宣言するには次のように書きます:

```
C_TEXT(myText)
```

いくつかの基本的な変数宣言の例です:

```
C_BLOB(vxMyBlob) // プロセス変数 vxMyBlob を BLOB型として宣言します C_DATE($vdCurDate) // ローカル変数 $vdCurDa  
C_LONGINT(vg1;vg2;vg3) // 3つのプロセス変数 vg1, vg2, vg3 を倍長整数型として宣言します  
C_OBJECT($vObj) // ローカル変数 $vObj をオブジェクト型として宣言します  
C_COLLECTION($vCol) // ローカル変数 $vCol をコレクション型として宣言します
```

注: 配列とは変数の一種で、同じタイプの変数を番号付きで並べたものです。配列は、配列宣言コマンド (例: `ARRAY LONGINT(a1AnArray;10)`) を使用して作成します。詳細については [配列](#) を参照ください。

変数への代入

変数を対象に、データを格納したり、格納したデータを別の対象にコピーしたりすることができます。変数にデータを格納することを、変数にデータを代入すると言い、代入演算子 (`:=`) を使っておこないます。代入演算子はフィールドに対してデータを代入する場合にも使います。

代入演算子は、変数を作成し、変数にデータを代入するために使用します。作成する変数名を代入演算子の左側に書きます。たとえば:

```
MyNumber:=3
```

は変数 `MyNumber` を作成し、数値 3を代入します。`MyNumber` が既に存在していれば、そこに数値 3が代入されます。

データ型の宣言 をせずに変数を作成することは通常推奨されません。

もちろん、変数からデータを取り出すことができなければ、便利とはいえません。再度代入演算子を使用します。`[Products]Size` というフィールドに `MyNumber` 変数の値を代入するには、代入演算子の右側に `MyNumber` を書きます:

```
[Products]Size:=MyNumber
```

これで、`[Products]Size` の値は3になります。この例はとても単純ですが、ある場所から別の場所へランゲージによってデータを転送させる基本的な手順を表しています。

配列要素にデータを代入するには中カッコ (`{...}`) を使用します:

```
atNames{1}:="Richard"
```

ローカル、プロセス、およびインタープロセス変数

ローカル、プロセス、および インターパロセス という、3種類の変数の変数を作成することができます。これらの変数の違いは使用できるスコープにあります。また、それらを使用することのできるオブジェクトも異なります。

ローカル変数

ローカル変数はその名のとおりメソッド内でローカルであり、変数が作成されたメソッドの範囲内でのみ使用可能で、その他のメソッドからはアクセスできません。メソッド内でローカルであるというのは、正式には「スコープがローカルである」といいます。ローカル変数は、その使用範囲をメソッド内に限定するために用います。

ローカル変数は、以下のような目的のために使用されます:

- 他の変数名との重複を避ける。
- データを一時的に使用する。

- プロセス変数の数を減らす。

ローカル変数の名前は必ずドル記号 (\$) で始め、この記号を除く31文字までの文字を指定できます。これより長い名前を指定すると、4D は余分の32文字以降を切り捨てます。

多くのメソッドや変数を持つアプリケーションプロジェクトで作業する場合、現在作業しているメソッドの範囲内で一時的に変数が必要となる場合がよくあります。この場合、同じ変数名が他で使用されていないかどうかを気にすることなくローカル変数を作成することができます。

アプリケーションではしばしば、ユーザーによる少量のデータ入力を必要とする場合があります。Request コマンドを使って、この情報を取得することができます。このコマンドはデータ入力を求めるダイアログボックスを表示し、ユーザーがデータを入力すると、その情報を戻り値として返します。このようなデータは通常、メソッド内で長期間維持する必要はありません。これは、ローカル変数を使用する典型的な例といえます。次に例を示します：

```
$vsID:=Request("ID を入力してください:")
If(OK=1)
    QUERY([People];[People]ID =$vsID)
End if
```

このメソッドは、ユーザーに ID を入力するように要求します。ローカル変数 \$vsID にレスポンスが代入され、ユーザーが入力した ID に基づいた検索がおこなわれます。このメソッドが終了した時点で、\$vsID ローカル変数はメモリから消去されます。この変数は 1回のみ、このメソッド内でしか使われないため、これ以上維持する必要はありません。

注：メソッドに渡される \$1, \$2... 等の引数はローカル変数です。詳細については [パラメーター](#) を参照ください。

プロセス変数

プロセス変数は、同じプロセスの範囲内に限り使用可能です。この変数はプロセスマソッドと、そのプロセス内で呼び出された他のメソッドで使用することができます。

プロセス変数には名前に付ける接頭辞がありません。プロセス変数の名前は、最大31文字までの長さで指定できます。

インターフォーマー・モードでは、変数は動的にメモリ上に作成・消去されます。これに対してコンパイルモードでは、作成したすべてのプロセス（ユーザー・プロセス）で同じプロセス変数定義が共有されますが、変数のインスタンスはプロセス毎に異なるものとなります。たとえば、プロセスP_1 とプロセスP_2 の両方にいてプロセス変数 myVar が存在していても、それらはそれぞれ別のインスタンスです。

バージョン6より、GET PROCESS VARIABLE や SET PROCESS VARIABLE を使用して、あるプロセスから他のプロセスのプロセス変数の値を取得したり、設定したりできるようになりました。これらのコマンドの利用は、以下のような状況に限定することが、良いプログラミングの作法です：

- コード内の特定の箇所におけるプロセス間通信
- プロセス間のドラッグ & ドロップ処理
- クライアント/サーバーにおいて、クライアントマシン上のプロセスとサーバーマシン上のストアドプロシージャー間の通信

詳細については [プロセス](#) の章と、各コマンドの説明を参照ください。

インターフォーマー・プロセス変数

インターフォーマー・プロセス変数はプロジェクト全体で使用することができ、すべてのコオペレティブ・プロセスで共有されます。これらは主としてプロセス間で情報を共有するために使われます。

プリエンプティブ・プロセスにおいては使用できないことと、コードの保守管理を煩雑にすることから、インターフォーマー・プロセス変数の使用は推奨されません。

インターフォーマー・プロセス変数の名前は、必ずインターフォーマー・プロセス記号 (<>) で始めます。記号の後に31バイトまでの名前を指定できます。

クライアント/サーバーでは、各マシン（クライアントマシンとサーバーマシン）で同じインターフォーマー・プロセス変数定義を共有しますが、マシンごとに各変数のインスタンスが存在します。

配列

配列とは、同じタイプの変数を番号付きで並べたものです。各変数は、配列の要素といいます。配列のサイズとは、配列が持つ要素の数を指します。配列は作成時にサイズが与えられ、要素の追加・挿入・削除によって、または作成時に使用したコマンドの再使用によって、何度もサイズを変更することができます。配列要素には、1からNの番号が付けられます(Nは配列のサイズ)。配列は必ず、特別な要素ゼロを持ちます。配列は4Dの変数です。他の変数と同様、配列にもスコープがあり、4D ランゲージの規則に従いますが、他と異なるところがいくつかあります。

ほとんどの場合において、配列よりコレクションの利用が推奨されます。コレクションは配列より柔軟だけでなく、たくさんの専用メソッドを持ちます。詳細については、[コレクション](#)を参照してください。

配列の作成

配列は、"配列"テーマの配列宣言コマンドのいずれかを使用して作成します。配列宣言コマンドは、1次元または2次元の配列の作成やサイズ変更を作ることができます。2次元配列の詳細については[二次元配列](#)を参照してください。

次のコードは、10個の要素からなる整数配列を作成(宣言)します:

```
ARRAY INTEGER(aiAnArray;10)
```

次のコードは、さきほど作成した配列を20要素にサイズ変更します:

```
ARRAY INTEGER(aiAnArray;20)
```

次のコードは、この配列を要素なしにサイズ変更します:

```
ARRAY INTEGER(aiAnArray;0)
```

配列要素への値の代入

配列中の要素は中カッコ({…})を使用して参照します。中カッコの中には数字を入れて特定の要素を指定します。この数字を要素番号といいます。次のコードは、5つの名前をatNamesという配列に入れ、それらを警告ウインドウに表示します:

```
ARRAY TEXT(atNames;5)
atNames{1}:="Richard"
atNames{2}:="Sarah"
atNames{3}:="Sam"
atNames{4}:="Jane"
atNames{5}:="John"
For($v1Elem;1;5)
    ALERT("要素番号 #"+String($v1Elem)+" の値は "+atNames{$v1Elem}+" です。")
End for
```

atNames{\$v1Elem}というシンタックスに注目してください。atNames{3}のように数値リテラルを使うだけでなく、数値変数によって配列の要素番号を指定することができます。ループ構造による反復を使用すると(For...End for, Repeat...Until または While...End while)、短いコードで配列の全要素、または一部の要素を対象とした処理をおこなうことができます。

重要: 代入演算子(:=)と比較演算子(=)とを混同しないように注意してください。代入と比較は、まったく異なる性質の処理です。

配列への配列の代入

文字列やテキスト変数と違って、配列に配列を代入することはできません。配列をそっくりそのまま別の配列にコピーするには COPY ARRAY コマンドを

を使います。

配列の要素ゼロ

配列は必ず、要素ゼロを持ちます。ドロップダウンリストなどのフォームオブジェクトに配列が設定されていた場合、要素ゼロが表示されることはできませんが、ランゲージでの利用に制限はありません（*）。

例として、デフォルト値を指定せずにフォームオブジェクトを初期化したいとします。このような場合に配列の要素ゼロが利用できます：

```
// atName 配列と紐づいているコンボボックスまたはドロップダウンリストの
// フォームオブジェクトメソッドです
Case of
  :(Form event code=On Load)
  // 要素ゼロを含め
  // 配列を初期化します
    ARRAY TEXT(atName;5)
    atName{0}:=選択してください"
    atName{1}:="Text1"
    atName{2}:="Text2"
    atName{3}:="Text3"
    atName{4}:="Text4"
    atName{5}:="Text5"
  // 配列の選択要素を要素ゼロに設定します
  atName:=0
End case
```

(*) ひとつだけ例外があります。配列タイプのリストボックスでは、編集中の元の値を保持するため、内部的に配列の要素ゼロが使用されます。この特別なケースでは、開発者は 0番目の要素を利用できません。

二次元配列

配列宣言コマンドはそれぞれ、1次元および 2次元の配列を作成、またはサイズ変更ができます。例：

```
ARRAY TEXT(atTopics;100;50) // 100行と 50列からなるテキスト配列を作成します
```

2次元配列は、本質的にはランゲージオブジェクトであり、表示や印刷することはできません。

上のコードで作成した atTopics 配列について、次のことが言えます：

- atTopics は、2次元配列です。
- atTopics{8}{5} は、8行5列目の要素です。
- atTopics{20} は 20行目を指し、それ自体が 1次元の配列です。
- `Size of array(atTopics)` は、行数の 100を返します。
- `Size of array(atTopics{17})` は、17行目の列数である50を返します。

以下の例では、データベースの各テーブルの各フィールドへのポインターが 2次元配列に格納されます：

```

C_LONGINT($vlLastTable;$vlLastField)
C_LONGINT($vlFieldNumber)
// テーブルと同じ数の空行（つまり、列なし）を持つ配列作成します
$vlLastTable:=Get last table number
ARRAY POINTER(<>apFields;$vlLastTable;0) // X行 0列の 2D配列
// テーブル毎に
For($vlTable;1;$vlLastTable)
  If(Is table number valid($vlTable))
    $vlLastField:=Get last field number($vlTable)
  // 全フィールドをチェックします
    $vlColumnNumber:=0
    For($vlField;1;$vlLastField)
      If(Is field number valid($vlTable;$vlField))
        $vlColumnNumber:=$vlColumnNumber+1
    // 当該テーブルの行にフィールドに対応する列を挿入していきます
      INSERT IN ARRAY(<>apFields{$vlTable};$vlColumnNumber;1)
  // 作成した "セル" にポインターを割り当てます
    <>apFields{$vlTable}{$vlColumnNumber}:=Field($vlTable;$vlField)
  End if
End for
End if
End for

```

このように初期化された 2次元配列を使って、以下の方法で特定のテーブルが持つ全フィールドへのポインターを取得できます：

```

// 現在選択されているテーブルの、フィールドへのポインターを取得します：
COPY ARRAY(<>apFields{Table(Current form table)};$apTheFieldsIamWorkingOn)
// ブールと日付フィールドを初期化します
For($vlElem;1;Size of array($apTheFieldsIamWorkingOn))
  Case of
    :(Type($apTheFieldsIamWorkingOn{$vlElem})=Is date)
      $apTheFieldsIamWorkingOn{$vlElem}:=:Current date
    :(Type($apTheFieldsIamWorkingOn{$vlElem})=Is Boolean)
      $apTheFieldsIamWorkingOn{$vlElem}:=:True
  End case
End for

```

注：この例でわかるように、2次元配列の行の列数はそれぞれが同じサイズでも異なるサイズでも構いません。

配列とメモリ

テーブルやレコードを使用してディスク上に格納するデータと異なり、配列は常に全体がメモリに保持されます。

たとえば、米国内の郵便番号がすべて [Zip Codes] テーブルに入力されている場合、約100,000件のレコードになります。加えて、そのテーブルには郵便番号のほかに、対応する市・郡・州という複数のフィールドがあるとします。カリフォルニアの郵便番号を選択した場合、4D データベースエンジンは [Zip Codes] テーブルから該当するレコードセクションを作成して、必要な場合にのみ各レコードをロードします（たとえば表示や印刷時）。つまり、4D のデータベースエンジンによってディスクからメモリに部分的にロードされた（フィールドごとに同じタイプの）順序づけられた一連の値で作業するということです。

同じことを配列で実行するのは、次の理由で禁止すべきです：

- 4つの情報タイプ（郵便番号、市、郡、州）を維持するためには、4つの大きな配列をメモリ内で維持する必要があります。
- 配列は、常に全体がメモリ内に維持されるため、常時使用しない場合でも、作業セッションの間すべてのデータをメモリに置いておく必要があります。
- 配列全体が常にメモリ内に維持されることから、アプリケーションが開始されるたびに 4つの配列をディスクからロードして、終了時にはディスクに保存する必要があります。当該データが作業セッション中に使用・変更されない場合もこれを省略することができません。

結論：配列は、ほどよい量のデータを短時間維持するためのものです。他方、配列はメモリ内に置かれるため、扱いやすく高速操作が可能です。

しかし、状況によっては何百、何千という要素を持った配列で作業する必要があります。次の表に、各配列タイプがメモリ上に占めるバイト数を求めるための計算式を示します：

配列タイプ	メモリ使用量の計算式 (バイト単位)
BLOB	(1+要素数) * 12 + 全BLOB要素の合計サイズ
ブール	(31+要素数) ¥ 8
日付	(1+要素数) * 6
整数	(1+要素数) * 2
倍長整数	(1+要素数) * 4
オブジェクト	(1+要素数) * 8 + 全オブジェクトの合計サイズ
ピクチャー	(1+要素数) * 8 + 全ピクチャーの合計サイズ
ポインター	(1+要素数) * 8 + 全ポインターの合計サイズ
実数	(1+要素数) * 8
テキスト	(1+要素数) * 20 + (全テキストの合計サイズ) * 2
時間	(1+要素数) * 4
2次元	(1+要素数) * 16 + 配列サイズの合計

注:

- メモリ中のテキストサイズは以下の式で計算されます: ((Length + 1) * 2)
- 選択した要素や要素数、配列自体の情報を保持するため、さらに数バイトを要します。

引数

メソッドや関数にデータを渡す必要がしばしば発生します。これは引数によって容易にできます。

概要

引数 (または パラメーター) とは、メソッドや関数が処理に必要とするデータのことです。引数とパラメーターは厳密には違うものですが、このマニュアルでは同義語として使用されています。引数は、ビルトインの 4Dコマンドにも渡されます。以下の例は、“Hello”という文字列を引数としてビルトインの ALERT コマンドへ渡します：

```
ALERT("Hello")
```

メソッドやクラス関数に引数を渡す場合も同様におこないます。たとえば、`getArea()` クラス関数が 2つの引数を受け取る場合、このクラス関数を呼び出すには以下のように書きます：

```
$area:=$o.getArea(50;100)
```

また、プロジェクトメソッド `DO SOMETHING` が3つの引数を受け取る場合、このメソッドを呼び出すには以下のように書きます：

```
DO_SOMETHING($WithThis;$AndThat;$ThisWay)
```

入力引数は、セミコロン (;) で区切れます。

メソッドを実行する専用コマンドを利用するときも、同じ原則で引数を渡します。

```
EXECUTE METHOD IN SUBFORM("Cal2";"SetCalendarDate";*;!05/05/20!)  
// サブフォーム "Cal2" のコンテキストにおいて SetCalendarDate を実行し  
// その際に引数として日付リテラル !05/05/20! を渡します
```

メソッドやクラス関数からデータを返すこともできます。以下は、文字列のデータ長を返すビルトインの `Length` コマンドを用いたステートメントです。このステートメントでは、`Length` 関数が `MyLength` という変数に値を返します。

```
MyLength:=Length("How did I get here?")
```

どのようなサブルーチンでも値を返すことができます。各メソッドやクラス関数につき、定義できる戻り値は一つだけです。

入力および出力値は呼び出し時に評価され、その値はそれぞれ自動的にサブルーチン (呼び出されたメソッドまたはクラス関数) 内のローカル変数に格納されます。呼び出されるメソッドにおいて、これらのローカル変数を宣言するには次の 2つのシンタックスが利用できます：

- **名前付き変数** (ほとんどの場合に推奨)
- **受け渡し順に番号が付けられた変数** (順番引数)

引数の宣言にあたって、**名前付き** シンタックスと **順番** シンタックスは制限なく併用することができます。たとえば：

```
Function add($x : Integer)  
  var $0;$2 : Integer  
  $0:=$x+$2
```

名前付き引数

呼び出されたメソッドやクラス関数において、引数の値はローカル変数に代入されます。引数は パラメーター名 とその データ型 をコロン (:) で区切って宣言することができます。

- クラス関数の場合、引数は `Function` キーワードとともに宣言されます。
- メソッドの場合（プロジェクトメソッド、フォームオブジェクトメソッド、データベースメソッド、トリガー）、引数はメソッドコード先頭の `#DECLARE` キーワードを使って宣言されます。

例:

```
Function getArea($width : Integer; $height : Integer) -> $area : Integer
```

```
// myProjectMethod  
#DECLARE ($i : Integer) -> $myResult : Object
```

次のルールが適用されます:

- 宣言文はメソッドや関数のコードの先頭に位置していなければなりません。宣言文より前に置けるのはコメントと改行のみであり、それ以外の場合はエラーが表示されます。
- 引数名は必ず \$ 文字で始まり、[プロパティ名の命名規則](#) に準拠している必要があります。
- 複数のパラメーター（およびその型）を宣言する場合は、それらをセミコロン (;) で区切れます。
- 複数行シンタックスがサポートされています ("¥" 文字を使用)。

たとえば、`getArea()` 関数に 2つの引数を渡して呼び出す場合:

```
$area:=$o.getArea(50;100)
```

クラス関数において、引数の値はそれぞれ対応するパラメーターに代入されます:

```
// クラス: Polygon  
Function getArea($width : Integer; $height : Integer)-> $area : Integer  
    $area:=$width*$height
```

パラメーターの型が宣言されていない場合には、[Variant](#) 型として定義されます。

データベースメソッドを含むすべての 4Dメソッドにおいて `#DECLARE` キーワードの使用がサポートされています。たとえば、`On Web Authentication` データベースメソッドにおいて、次のように名前付き引数を宣言できます:

```
// On Web Authentication データベースメソッド  
#DECLARE ($url : Text; $header : Text; \  
    $BrowserIP : Text; $ServerIP : Text; \  
    $user : Text; $password : Text) \  
    -> $RequestAccepted : Boolean  
$entitySelection:=ds.User.query("login=:1"; $user)  
// ハッシュパスワードを確認...
```

戻り値

関数の戻り値は、入力パラメータリストに矢印 (->) を追加し、それに続けて宣言します。たとえば:

```
Function add($x : Variant; $y : Integer) -> $result : Integer
```

矢印と出力変数名を省略して、コロン (:) 記号の後に戻り値のデータ型だけを指定した場合には、戻り値は [return文](#) または [受け渡し順シンタックス](#) の `$0` を使って管理します。たとえば:

```
Function add($x : Variant; $y : Integer): Integer  
$0:=$x+$y
```

サポートされているデータ型

名前付き引数の場合、`var` キーワードでサポートされている データ型 (クラスオブジェクト含む) を使用できます。たとえば:

```
Function saveToFile($entity : cs.ShapesEntity; $file : 4D.File)
```

順番引数

名前付き引数 シンタックスを使用するほかにも、引数は受け渡し順に番号が付けられた変数を使って宣言することができます: `$1, $2, $3, ...`。ローカル変数の番号は、引数の順番を表わします。

このシンタックスはクラス関数の場合もサポートされていますが、名前付き引数 を使ったシンタックスの方が推奨されます。

たとえば、プロジェクトメソッド `DO SOMETHING` が3つの引数を受け取る場合、このメソッドを呼び出すには以下のように書きます:

```
DO_SOMETHING($WithThis;$AndThat;$ThisWay)
```

呼び出されるメソッドにおいて、それぞれの引数の値は自動的に、順に番号が付けられたローカル変数 (`$1, $2, $3...`) に格納されます:

```
// DO_SOMETHING メソッド  
// すべての引数はテキスト型です  
C_TEXT($1;$2;$3)  
ALERT($1+" と "+$2+" と "+$3+" を受け取りました。")  
// $1 には $WithThis の値が代入されます  
// $2 には $AndThat の値が代入されます  
// $3 には $ThisWay の値が代入されます
```

戻り値

戻り値は自動的に、ローカル変数 `$0` に格納します。

たとえば、`Uppercase4` という以下のメソッドは、始めの 4 文字を大文字に変換した文字列を返します:

```
// Uppercase4 メソッド  
$0:=Uppercase(Substring($1;1;4))+Substring($1;5)
```

以下は、`Uppercase4` をメソッドとして使用する例です:

```
$NewPhrase:=Uppercase4("This is good.")
```

変数 `$NewPhrase` には "THIS is good." が格納されます。

戻り値 `$0` はサブルーチン内のローカル変数です。したがって、サブルーチン内で通常のローカル変数のように使用できます。たとえば:

```
// Do_something メソッド  
$0:=Uppercase($1)  
ALERT($0)
```

この例において、`$0` は大文字に変換した引数 `$1` の値を割り当てられ、その後 `ALERT` コマンドに引数として渡されました。このように、サブルーチン内の他のローカル変数と同じように `$0` を使うことができます。サブルーチン終了時に、その時点での `$0` の値を呼び出し元のメソッドに戻すのは 4Dがおこないます。

サポートされているデータ型

順番引数には、あらゆる [式](#) の形が使用できますが、例外があります：

- `tables`
- `arrays`

テーブルや配列の式は [ポインターを介した参照として](#) 渡す必要があります。

`return {expression}`

▶ 履歴

`return` 文は、関数やメソッドの実行を終了させ、呼び出し元に式を返すために使用します。

たとえば、次の関数は引数 `$x` の 2乗を返します。`$x` 数値です。

```
Function square($x : Integer)
    return $x * $x
```

内部的に、`return x` は `$0:=x` または（宣言されていれば）`myReturnValue:=x` を実行し、呼び出し元に戻ります。`return` が式なしで使われた場合、関数またはメソッドは宣言された戻り値の型（あれば）の `null` 値を返し、それ以外の場合には `undefined` です。

`return` 文は、[戻り値](#) の標準的なシンタクスと併用することができます（戻り値は宣言された型でなくてはなりません）。ただし、`return` はコードの実行を直ちに終了させることに注意が必要です。たとえば：

```
Function getValue
    $0:=10
    return 20
    // 20 が返されます

Function getValue -> $v : Integer
    return 10
    $v:=20 // 実行されません
    // 10 が返されます
```

引数の間接参照 (`${N}`)

4Dプロジェクトメソッドは、可変個の引数を受け取ることができます。`For...End for` ループや `Count parameters` コマンド、引数の間接参照シンタクスを使って、これらの引数を扱うことができます。メソッド内で、間接参照は `${N}` のように表示します。この `N` は数値式です。 `${N}` を ジェネリックパラメーター (generic parameter) と呼びます。

ジェネリックパラメーターの使い方

以下は間接参照の例です。引数の数値を合計した結果を、引数として渡された表示形式で返すようなメソッドを考えてみましょう。合計される数値の数は、メソッドが呼ばれるたびに変わります。このメソッドでは数値と表示形式を引数としてメソッドに渡さなければなりません。

以下は `MySum` メソッドです：

```
#DECLARE($format : Text) -> $result : Text
$sum:=0
For($i;2;Count parameters)
    $sum:=$sum+${$i}
End for
$result:=String($sum;$format)
```

このメソッドの引数は正しい順序で渡す必要があります。最初に表示形式、次に可変個の数値引数です。

```
Result:=MySum("##0.00";125,2;33,5;24) // "182.70"
Result:=MySum("000";1;2;200) // "203"
```

メソッド内で 0、1、またはそれ以上のパラメーターを宣言した場合でも、任意の数の引数を渡すことができます。呼び出されたメソッド内では、 `${N}` シンタックスを使って引数を利用でき、可変長引数の型はデフォルトで **バリアント** です（コンパイラー指示子 を使ってこれらを宣言できます）。`Count parameters` コマンドを使用して、パラメーターが存在することをあらかじめ確認しておく必要があります。たとえば：

```
// foo メソッド
#DECLARE($p1: Text;$p2 : Text; $p3 : Date)
For($i;1;Count parameters)
    ALERT("param "+String($i)+" = "+String(${$i}))
End for
```

このメソッドは次のように呼び出せます：

```
foo("hello","world";!01/01/2021!;42;?12:00:00?) // 追加の引数が受け渡されます
```

引数の間接参照は以下の条件を守ることにより、正しく動作します：引数の一部のみを間接参照する場合、直接参照する引数の後に間接参照引数を配置するようにします。

ジェネリックパラメーターの宣言

他のローカル変数と同様、ジェネリックパラメーターはコンパイラーに指示する必要はありません。ただし、曖昧さを回避するためには推奨されます。宣言なしのジェネリックパラメーターは自動的に **Variant** 型となります。

ジェネリックパラメーターの宣言には、コンパイラー指示子に `${N}` を渡す、以下のシンタックスを使用します（N は 1つ目の最初のジェネリックパラメーターの番号です）：

```
C_TEXT(${4})
```

ジェネリックパラメーターの宣言は **受け渡し順** シンタックスでのみ使用できます。

このコマンドは、4番目以降に間接参照されるすべての引数のデータ型がテキストであることを意味します。`$1`、`$2`、`$3`には、いかなるデータ型も使用できますが、`$2`を間接参照した場合には、間接参照の型宣言の影響を受けます。このため、たとえば `$2` が実数であっても、間接参照されればテキストと見なされます。

宣言に使用する数値は変数ではなく、定数でなくてはなりません。

コンパイルモード用のパラメーター宣言

[インタープリターモード](#) では必須ではないものの、問題を避けるにはメソッドや関数の各パラメーターを宣言しておくべきでしょう。

名前付き引数シンタックス を利用している場合には、それらの引数は `#DECLARE` キーワードまたは `Function` プロトタイプによって自動的に宣言されます。たとえば:

```
Function add($x : Variant; $y : Integer) -> $result : Integer
// すべての引数はデータ型とともに宣言されます
```

順番引数シンタックス を利用している場合には、引数がそれぞれ適切に宣言されていることを確認する必要があります。次の例では `Capitalize` プロジェクトメソッドは第1パラメーターにテキスト型の引数を受け取り、戻り値としてテキスト型の値を返します:

```
// Capitlize プロジェクトメソッド
// Capitlize ( Text ) -> テキスト
// Capitlize ( Source string ) -> 大文字の文字列

C_TEXT($0;$1)
$0:=Uppercase(Substring($1;1;1))+Lowercase(Substring($1;2))
```

`New process` コマンドなどでプロセスメソッドを呼び出す場合にも、そのメソッドが引数を受け取るのであれば、それらは明示的に宣言されていなくてはなりません。たとえば:

```
C_TEXT($string)
C_LONGINT($idProc;$int)
C_OBJECT($obj)

$idProc:=New process("foo_method";0;"foo_process";$string;$int;$obj)
```

"foo_method" において各パラメーターが適切に宣言されている場合のみ、コンパイルモードで上のコードを実行することができます:

```
//foo_method
C_TEXT($1)
C_LONGINT($2)
C_OBJECT($3)

...
```

プロジェクトメソッドのパラメーター宣言は、コンパイルモード用にまとめて、"Compiler" で始まる名称の専用メソッドにておこなうことができます。専用メソッド内で各メソッドのパラメーターをあらかじめ宣言する場合は、次のように書きます:

```
// Compiler_method
C_REAL(OneMethodAmongOthers;$1)
```

詳細については [インターフィラモードとコンパイルモード](#) を参照ください。

パラメーターの宣言は次のコンテキストにおいても必須となります (これらのコンテキストは "Compiler" メソッドによる一括宣言をサポートしません)。

- データベースメソッド - たとえば、`On Web Connection` データベースメソッド は 6つのテキスト型の引数 `$1 ~ $6` を受け取ります。たとえすべての引数を使用しない場合でも、データベースメソッドの先頭で次のように宣言しなくてはなりません:

```
// On Web Connection
C_TEXT($1;$2;$3;$4;$5;$6)
```

`#DECLARE` キーワードを使用して、[名前付き引数](#) を使うこともできます。

- トリガー - トリガーの結果である `$0` パラメーター (倍長整数) は、明確に定義されなければコンパイラによって型指定されます。定義する場合は、トリガーの中でおこなう必要があります。

- On Drag Over フォームイベントを受け入れるフォームオブジェクト - On Drag Over フォームイベントの結果である \$0 パラメーター (倍長整数) は、明確に定義されなければコンパイラが型を決定します。定義する場合は、オブジェクトメソッドの中でおこなう必要があります。注: コンパイラは \$0 を初期化しません。したがって、On Drag Over フォームイベントを使用したら、直ちに \$0 を初期化しなければなりません。たとえば:

```
C_LONGINT($0)
If(Form event=On Drag Over)
    $0:=0
    ...
    If($DataType=Is picture)
        $0:=-1
    End if
    ...
End if
```

引数の型間違い

間違った型の引数を呼び出すことは、正しい実行を妨げる エラー となります。たとえば、次のようなメソッドを書いたとします:

```
// メソッド1
#DECLARE($value : Text)
```

```
// メソッド2
method1(42) // 型間違い。期待されるのはテキスト
```

このケースは、コンテキストに応じて 4D で処理されます。

- コンパイル済みプロジェクト では、可能な限りコンパイル時にエラーが生成されます。それ以外の場合は、メソッドの呼び出し時にエラーが生成されます。
- インタープリタープロジェクトでは:
 - 名前付きシンタックス (#DECLARE または Function) を使用して引数が宣言されている場合は、メソッドの呼び出し時にエラーが発生します。
 - 順番シンタックス (C_XXX) を使用して宣言されている場合、エラーは発生せず、呼び出されたメソッドは期待される型の空の値を受け取ります。

入力 / 出力変数

これらの引数 (\$1, \$2...) はサブルーチン内で他のローカル変数と同様に使用できます。しかしながら、引数として渡した変数の値を変更するコマンドをサブルーチン内で使用する場合 (例: Find in field)、\$1, \$2などを直接渡すことはできません。まず標準のローカル変数等にコピーする必要があります (例: \$myvar:=\$1)。

オブジェクトプロパティを名前付き引数として使用する

引数としてオブジェクトを渡すことによって 名前付き引数 を扱うことができます。このプログラミング方法はシンプルかつ柔軟なだけでなく、コードの可読性も向上させます。

たとえば、CreatePerson メソッドを例にとると:

```
// CreatePerson メソッド
var $person : Object
$person:=New object("Name";"Smith";"Age";40)
ChangeAge($person)
ALERT(String($person.Age))
```

ChangeAge メソッドを次のように書けます:

```
// ChangeAge メソッド
var $1; $para : Object
$para:=$1
$para.Age:=$para.Age+10
ALERT($para.Name+" は "+String($para.Age)+" 歳です。")
```

これは [任意パラメーター](#) を指定するにあたって非常に便利な方法です (後述参照)。この場合、引数の不足は次のように対処できます:

- `Null` 値と比較することで、必要な引数がすべて提供されているかをチェックします
- 引数の値をプリセットします
- 渡されていない引数は空値として扱います

上述の ChangeAge メソッドの例では、Age およびName プロパティはどちらも必須であるため、引数オブジェクトに含まれていなければエラーが発生します。これを避けるには、次のように記述することができます:

```
// ChangeAge メソッド
var $1; $para : Object
$para:=$1
$para.Age:=Num($para.Age)+10
ALERT(String($para.Name)+" は "+String($para.Age)+"歳です。")
```

すると、引数が不足してもエラーは生成されず、両方が欠落した場合の結果は " is 10 years old" となってしまうにせよ、いずれの引数も任意となります。

名前付き引数を利用すると、アプリケーションの保守やリファクタリングが簡単かつ安全におこなえます。さきほどの例で、加算する年数を場合に応じて変えたほうが適切であると、あとから気づいたとします。メソッドのパラメーターとして、加算年数を追加しなくてはなりません。この場合、次のように書けます:

```
$person:=New object("Name";"Smith";"Age";40;"toAdd";10)
ChangeAge($person)

// ChangeAge メソッド
var $1;$para : Object
$para:=$1
If ($para.toAdd=NULL)
    $para.toAdd:=10
End if
$para.Age:=Num($para.Age)+$para.toAdd
ALERT(String($para.Name)+" は "+String($para.Age)+" 歳です。")
```

このように、既存のコードを変える必要はありません。変更後のコードは変更前と同じように動作しますが、引数によって加算年数に数値を指定することもできるようになりました。

名前付き引数を使うと、すべてのパラメーターを任意にすることができます。上の例ではすべてのパラメーターが任意で、いずれを指定しても順序はありません。

任意パラメーター

4D ランゲージリファレンスにおいて、コマンドシンタックス中の { } 文字 (中括弧) はその引数が省略可能であることを示します。たとえば、`ALERT(message{}; okButtonTitle{})` は `okButtonTitle` が省略できることを意味します。この場合、次のような呼び出し方が可能です:

```
ALERT("Are you sure?";"Yes I am") // 2つの引数
ALERT("Time is over") // 1つの引数
```

4Dメソッドや関数も、このような任意パラメーターを受け入れます。任意の数のパラメーターを宣言することができます。宣言されているよりも少ない引数をメソッドや関数に渡した場合、指定されなかったパラメーターは、[そのタイプに応じたデフォルト値](#) として、呼び出されたコードの中で処理されます。たとえば:

```
// myClass クラスの "concat" 関数
Function concat ($param1 : Text ; $param2 : Text)->$result : Text
$result:=$param1+" "+$param2
```

```
//呼び出し元メソッド
$class:=cs.myClass.new()
$class.concat("Hello") // "Hello "
$class.concat() // スペースのみ: "
```

宣言されているよりも多い数のパラメーターをメソッドや関数に渡すこともできます。呼び出されたコードにおいて、これらは [\\${N} シンタックス](#) を使うことで利用可能です。

Count parameters コマンドを使用すると、メソッドに渡された引数の数を確認することができるため、数に応じて異なる処理をおこなえます。

次の例はテキストメッセージを表示し、2つの引数が渡されていればディスク上のドキュメントに、3つ以上の場合には 4D Write Pro エリアにそのテキストを書き出します。

```
// APPEND TEXT プロジェクトメソッド
// APPEND TEXT ( テキスト { ; テキスト { ; オブジェクト } } )
// APPEND TEXT ( メッセージ { ; パス { ; 4DWPエリア } } )

Method($message : Text; $path : Text; $wpArea : Object)

ALERT($message)
If(Count parameters>=3)
    WP SET TEXT($wpArea;$1;wk append)
Else
    If(Count parameters>=2)
        TEXT TO DOCUMENT($path;$message)
    End if
End if
```

このプロジェクトメソッドをアプリケーションに追加したあとは、次のように呼び出すことができます：

```
APPEND TEXT(vtSomeText) // メッセージを表示します
APPEND TEXT(vtSomeText;$path) // メッセージを表示して、$path のドキュメントに書き出します
APPEND TEXT(vtSomeText;"";$wpArea) // メッセージを表示して、$wpArea の4D Write Pro ドキュメントに追記します
```

任意パラメーターが必要な場合、[オブジェクトプロパティを名前付き引数として使用する](#)と型の制限がなく、柔軟で便利です。

引数の渡し方：値か参照か

引数を渡すとき、4D は呼び出し元メソッドのコンテキストにおいてその式を評価し、結果の値をクラス関数またはサブルーチンのローカル変数に格納します。これらのローカル変数に格納されているのは、呼び出し元で使用されているフィールドや変数、式ではなく、渡された値のみです。スコープがローカルに限られているため、クラス関数 / サブルーチン内でローカル変数の値を変えても、呼び出し元メソッドには影響ありません。たとえば：

```
// MY_METHOD メソッド
DO_SOMETHING([People]Name) // [People]Name の値が "williams" だとします
ALERT([People]Name)

// DO_SOMETHING メソッド
$1:=Uppercase($1)
ALERT($1)
```

`DO_SOMETHING` メソッドによって表示されたアラートボックスでは "WILLIAMS" と表示され、`MY_METHOD` メソッドによって表示されるアラートボックスでは "williams" と表示されます。`DO_SOMETHING` メソッドは \$1 の値をローカルな範囲で変更しましたが、これは `MY_METHOD` メソッドがサブルーチンに渡す引数として指定した [People]Last Name フィールドの値には影響しません。

もし `DO_SOMETHING` メソッド内でフィールドの値を変更したいのであれば、2通りのやり方があります：

1. サブルーチンに渡す式としてフィールドではなく、フィールドへのポインターを指定することができます。この場合、以下のようにコードを書きます：

```
// MY_METHOD メソッド
DO_SOMETHING(>[People]Name) // [People]Name の値が "williams" だとします
ALERT([People]Last Name)

// DO_SOMETHING メソッド
$1->:=Uppercase($1->)
ALERT($1->)
```

この例では、引数として指定された式はフィールドではなく、フィールドへのポインターです。そのため、`DO_SOMETHING` メソッド内において、\$1 はフィールドの値ではなく、フィールドへのポインターになっています。\$1 引数によって参照される対象（上記コード内の \$1->）はフィールドそのものです。その結果、参照されている対象を変更すると、その影響はサブルーチンのスコープを超えて、実際のフィールドも変更されます。さきほどの例題においては、両方のアラートボックスに "WILLIAMS" と表示されます。

2. `DO_SOMETHING` メソッドに "何かさせる" 代わりに、値を返すようにメソッドを書き直すこともできます。たとえば、以下のようにコードです：

```
// MY_METHOD メソッド
[People]Name:=DO_SOMETHING([People]Name) // もとの [People]Name の値が "williams" だとします
ALERT([People]Name)

// DO_SOMETHING メソッド
$0:=Uppercase($1)
ALERT($0)
```

このようにサブルーチンの戻り値を使うことを "関数を使う" と言います。詳細については [戻り値](#) の章を参照ください。

特殊ケース：オブジェクトやコレクションの場合

オブジェクトやコレクションのデータタイプは参照（つまり、内部的なポインター）を介した形でのみ扱われることに注意が必要です。

したがって、\$1、\$2... には 値 ではなく 参照 が格納されます。 \$1、\$2... の値をサブルーチン内で変更した場合、その変更は元となるオブジェクトやコレクションが使用されているところへと伝播します。これは [ポインター](#) に対する原理と同じものですが、\$1、\$2... の使用にあたって参照を外す必要はありません。

次の例では、`CreatePerson` メソッドはオブジェクトを作成したのち、それを引数として `ChangeAge` に渡します：

```
// CreatePerson メソッド
var $person : Object
$person:=New object("Name";"Smith";"Age";40)
ChangeAge($person)
ALERT(String($person.Age))
```

`ChangeAge` メソッドは受け取ったオブジェクトの Age 属性に 10を加えます：

```
// ChangeAge メソッド
#DECLARE ($person : Object)
$person.Age:=$person.Age+10
ALERT(String($person.Age))
```

`CreatePerson` メソッドを実行すると、サブルーチンにおいても同じオブジェクト参照が扱われているため、両方のアラートボックスにおいて "50" と表示されます。

4D Server: "サーバー上で実行" オプションが使用された場合など、同じマシン上で実行されないメソッド間で引数が渡される場合、参照渡しは利用できません。このような場合には、参照の代わりにオブジェクトとコレクションのコピーが引数として渡されます。

共有オブジェクトと共有コレクション

共有オブジェクト および 共有コレクション はプロセス間でコンテンツを共有することができる、特殊な オブジェクト と コレクション です。 インタープロセス変数 に比べると、共有オブジェクトと共有コレクションは プリエンプティブ4Dプロセスと互換性があるという点で利点があります。つまり、 New process や CALL WORKER といったコマンドの引数として、参照の形で渡すことができるということです。

共有オブジェクトと共有コレクションは、標準の C_OBJECT と C_COLLECTION コマンドで宣言された変数に保存することができますが、専用のコマンドを使用してインスタンス化されている必要があります：

- 共有オブジェクトを作成するには、 New shared object コマンドを使用します。
- 共有コレクションを作成するには、 New shared collection コマンドを使用します。

注：共有オブジェクトと共有コレクションは標準の（非共有の）オブジェクトおよびコレクションのプロパティとして設定することができます。

共有オブジェクト/コレクションを編集するには、 Use...End use 構文を使う必要があります。共有オブジェクト/コレクションの値を読むにあたっては、 Use...End use は必要ありません。

Storage コマンドが返す、データベースにおいて固有かつグローバルなカタログは、そのアプリケーション内あるいはコンポーネントからいつでも利用することができ、すべての共有オブジェクトおよびコレクションを保存するのに使用することができます。

共有オブジェクト/共有コレクションの使用

New shared object あるいは New shared collection コマンドでインスタンス化されると、その共有オブジェクト/コレクションの属性と要素はどのプロセスからでも編集/読み出しができるようになります。

編集

共有オブジェクトと共有コレクションは、編集することができます：

- オブジェクトプロパティの追加・削除
- 値の追加・編集（共有オブジェクトがサポートしている範囲内で）。これには、他の共有オブジェクトやコレクションの追加・編集も含まれます（この場合、共有グループを作成します。後述参照）

ただし、共有オブジェクトあるいは共有コレクションを編集するコードは、必ず Use...End use 構文に組み込まれている必要があります、そうでない場合にはエラーが返されます。

```
$s_obj:=New shared object("prop1";"alpha")
Use($s_obj)
  $s_obj.prop1:="omega"
End Use
```

一度に 1 プロセスのみ、共有オブジェクト/コレクションを編集することができます。 Use は共有オブジェクト/コレクションを他のスレッドからアクセスできないようにロックする一方、 End use はこのロックを解除します（ロックカウンターが 0 の場合；後述参照）。 Use...End use を使わずに共有オブジェクト/コレクションを編集しようとすると、エラーが生成されます。すでに他のプロセスによって使用されている共有オブジェクト/コレクションに対して、別のプロセスが Use...End use を呼び出した場合、先着プロセスが End use でロックを解除するまで、その呼び出しは待機状態になります（エラーは生成されません）。したがって、 Use...End use 構文内の処理は迅速に実行され、ロックは可及的速やかに解除される必要があります。そのため、共有オブジェクト/コレクションをインターフェース（ダイアログボックスなど）から直接編集することは避けることが強く推奨されます。

共有オブジェクト/コレクションを他の共有オブジェクト/コレクションのプロパティあるいは要素に割り当てるることは可能で、このとき 共有グループ が作成されます。共有グループは、共有オブジェクト/コレクションのプロパティ値あるいは要素として他の共有オブジェクト/コレクションが設定されたときに自動的に作成されます。共有グループを使用すると共有オブジェクトを入れ子にすることができますが、以下のルールに気をつける必要があります：

- あるグループの共有オブジェクト/コレクションに対して Use を使うと、そのグループに所属するすべての共有オブジェクト/コレクションのプロパティ/要素がロックされ、ロックカウンターを 1 増加されます。 End use はグループのロックカウンターを 1 減らします。カウンターが 0 になると、すべてのリンクされた共有オブジェクト/コレクションのロックが解除されます。
- 共有オブジェクト/コレクションは一つの共有グループにしか所属することができません。すでにグループに所属している共有オブジェクト/コレクションを他のグループへと割り当てようとした場合、エラーが返されます。
- 一旦グループ化された共有オブジェクト/コレクションについて、グループを解除することはできません。一度共有グループに含まれた共有オブジェクト/

コレクションは、セッション中はずっと同グループに所属することになります。親オブジェクト/コレクションから子オブジェクト/コレクションへの参照をすべて削除したとしても、両者のリンクが解除されるわけではありません。

共有グループのルールについての詳細は、例題2を参照してください。

注: 共有グループは、ロック識別子と呼ばれる内部プロパティによって管理されています。この値についての詳細は、[ランゲージリファレンス](#) を参照ください。

読み出し

たとえ共有オブジェクト/コレクションが他のプロセスによって使用中であっても、それらのプロパティや要素は、`Use...End use` 構文を呼び出さずとも取得することができます。

ただし、複数の値が互いにリンクしていてそれらを一度に読み出す必要がある場合には、一貫性の観点から、共有オブジェクト/コレクションを `Use...End use` 内で扱う必要があります。

複製

共有オブジェクト（あるいは共有オブジェクトをプロパティとして格納しているオブジェクト）に対して `OB Copy` コマンドを使用することは可能ですが、含まれている子オブジェクト含め、標準（非共有）のオブジェクトが戻り値として返されます。

ストレージ

ストレージは固有の共有オブジェクトで、各アプリケーションおよびマシン上で利用可能です。この共有オブジェクトは `Storage` コマンドから返されます。このオブジェクトは、他のプリエンティプあるいは標準プロセスからでも利用出来るように、セッション中に定義されたすべての共有オブジェクト/コレクションを参照するためのものです。

ストレージ オブジェクトは標準の共有オブジェクトとは異なり、共有オブジェクト/コレクションがプロパティとして追加されたときでも共有グループを作成しないという点に注意してください。この例外的な振る舞いにより、ストレージ オブジェクトを使用するたびに、リンクされている共有オブジェクト/コレクションをすべてロックせずに済みます。

詳細な情報については、`Storage` コマンドの詳細を参照してください。

Use...End use

`Use...End use` 構文の正式なシンタックスは、以下の通りです：

```
Use(Shared_object_or_Shared_collection)
    statement(s) // ステートメント
End use
```

`Use...End use` 構文は、内部セマフォーの保護下において `Shared_object_or_Shared_collection` 引数に対して処理を実行するステートメントを定義します。`Shared_object_or_Shared_collection` として任意の有効な共有オブジェクトあるいは共有コレクションを渡すことができます。

共有オブジェクトおよび共有コレクションは、プロセス間の（とくに プリエンティプ4Dプロセス間の）通信ができるように設計されています。これらはプロセスから他のプロセスへ、参照型の引数として渡すことができます。共有オブジェクトおよび共有コレクションについての詳細な情報については、共有オブジェクトと共有コレクションを参照してください。共有オブジェクトおよび共有コレクションを扱う際には、複数プロセスによる同時アクセスを避けるために、必ずそれを `Use...End use` キーワードでくる必要があります。

- `Use` の実行が成功すると、対応する `**End use` が実行されるまで、`Shared_object_or_Shared_collection` のすべてのプロパティ/要素は他のあらゆるプロセスに対し書き込みアクセスがロックされます。
- `statement(s)` で実行されるステートメントは、`Shared_object_or_Shared_collection` のプロパティ/要素に対して、競合アクセスのリスクなしに変更も実行することができます。
- `Shared_object_or_Shared_collection` に他の共有オブジェクトあるいはコレクションがプロパティとして追加された場合、それらも同じ共有グループとして連結されます（共有オブジェクト/共有コレクションの使用を参照してください）。
- `**Use...End use **` 内ステートメントの実行中に、他のプロセスが `Shared_object_or_Shared_collection` のプロパティやリンクされたプロパティにアクセスしようとした場合、そのアクセスは自動的に保留され、実行中の処理が終了するまで待機します。
- `End use` は、`Shared_object_or_Shared_collection` プロパティおよび、同じグループのすべてのオブジェクトのロックを解除します。
- 4D コード内では、複数の `Use...End use` 構文を入れ子にすることができます。グループの場合、`Use` を使用するごとにグループのロックカウンターが 1 増加し、`End use` ごとに 1 減少します。最後の `End use` によってロックカウンターが 0 になった場合にのみ、すべてのプロパティ/要素のロッ

クが解除されます。

注: コレクションのメンバーメソッドが共有コレクションを変更する場合、そのメソッド実行中は、対象の共有コレクションに対して Use が内部的に自動で呼ばれます。

例題 1

それぞれ異なる製品の在庫更新を実行する複数のプロセスを起動し、同じ共有オブジェクトを更新していきます。まずメインプロセスで空の共有オブジェクトをインスタンス化してから、共有オブジェクトへの参照と対象製品を引数として渡して別プロセス起動します:

```
ARRAY TEXT($_items;0)
...
... // 在庫を確認する製品を配列に格納します
$nbItems:=Size of array($_items)
C_OBJECT($inventory)
$inventory:=New shared object
Use($inventory)
    $inventory.nbItems:=$nbItems
End use

// プロセスを起動します
For($i;1;$nbItems)
    $ps:=New process("HowMany";0;"HowMany_"+$_items{$i};$_items{$i};$inventory)
    // $inventory オブジェクトは参照で渡されます
End for
```

"HowMany" メソッド内では、在庫確認が終わるとすぐに \$inventory 共有オブジェクトが更新されます:

```
C_TEXT($1)
C_TEXT($what)
C_OBJECT($2)
C_OBJECT($inventory)
$what:=$1 // 可読性のため
$inventory:=$2

$count:=CountMethod($what) // 在庫確認用のメソッド
Use($inventory) // 共有オブジェクトを使用します
    $inventory[$what]:=$count // 当該製品の在庫を保存します
End use
```

例題 2

以下の例題は、共有グループを扱う際のルールについて説明しています:

```
$ob1:=New shared object
$ob2:=New shared object
Use($ob1)
  $ob1.a:=$ob2 // グループ1 が作成されます
End use

$ob3:=New shared object
$ob4:=New shared object
Use($ob3)
  $ob3.a:=$ob4 // グループ2 が作成されます
End use

Use($ob1) // グループ1のオブジェクトを使用します
  $ob1.b:=$ob4 // これはエラーになります
  // $ob4 はすでに他のグループに所属しているため
  // 代入することはできません
End use

Use($ob3)
  $ob3.a:=Null // グループ2から$ob4 への参照をすべて解除します

End use

Use($ob1) // グループ1のオブジェクトを使用します
  $ob1.b:=$ob4 // これもエラーになります
  // $ob4 は依然としてグループ2に所属しているため
  // 代入は不可能です
End use
```

クラス

概要

4D ランゲージでは クラス の概念がサポートされています。プログラミング言語では、クラスを利用することによって、属性やメソッドなどを持つ特定のオブジェクト種を定義することができます。

ユーザークラスが定義されていれば、そのクラスのオブジェクトをコード内で インスタンス化 することができます。各オブジェクトは、それ自身が属するクラスのインスタンスです。クラスは、別のクラスを [継承](#) することで、その [関数](#) と、([スタティック](#) および [計算された](#)) プロパティを受け継ぐことができます。

4D におけるクラスモデルは JavaScript のクラスに類似しており、プロトタイプチェーンに基づきます。

たとえば、次のように `Person` クラスを定義した場合:

```
// クラス: Person.4dm
Class constructor($firstname : Text; $lastname : Text)
    This.firstName:=$firstname
    This.lastName:=$lastname

Function get fullName() -> $fullName : text
    $fullName:=This.firstName+" "+This.lastName

Function sayHello()->$welcome : Text
    $welcome:="Hello "+This.fullName
```

この "Person" のインスタンスをメソッド内で作成するには、以下のように書けます:

```
var $person : cs.Person // Person クラスのオブジェクト
var $hello : Text
$person:=cs.Person.new("John";"Doe")
// $person:{firstName: "John"; lastName: "Doe"; fullName: "John Doe"}
$hello:=$person.sayHello() // "Hello John Doe"
```

クラスの管理

クラス定義

4Dにおいてユーザークラスとは、`/Project/Sources/Classes/` フォルダーに保存された専用のメソッドファイル (.4dm) によって定義されます。ファイル名がクラス名になります。

クラスを命名する際には、次のルールに留意してください:

- [クラス名](#) は [プロパティ名の命名規則](#) に準拠している必要があります。
- クラス名の大文字・小文字は区別されます。
- 競合防止のため、データベースのテーブルと同じ名前のクラスを作成するのは推奨されないこと

たとえば、"Polygon" という名前のクラスを定義するには、次のファイルを作成する必要があります:

- Project フォルダー
 - Project

```
* Sources
  - Classes
    + Polygon.4dm
```

クラスの削除

既存のクラスを削除するには:

- ディスク上で "Classes" フォルダーより .4dm クラスファイルを削除します。
- 4D エクスプローラーでは、クラスを選択した状態で **-** をクリックするか、コンテキストメニューより 移動 > ゴミ箱 を選択します。

4D インターフェースの使用

ファイル メニューまたはエクスプローラーなど、4D インターフェースを介してクラスを作成した場合には、クラスファイルは自動的に適切な場所に保存されます。

ファイルメニューとツールバー

4D 開発の ファイル メニューまたはツールバーより 新規 > クラス... を選択することで、開いているプロジェクトにクラスファイルを新規作成することができます。

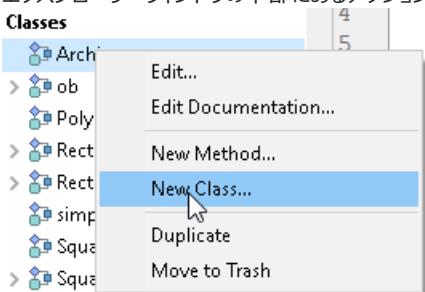
Ctrl+Shift+Alt+k ショートカットも使用できます。

エクスプローラー

エクスプローラーの メソッド ページにおいて、クラスは クラス カテゴリに分類されています。

クラスを新規作成するには次の方法があります:

- クラス カテゴリを選択し、**+** ボタンをクリックします。
- エクスプローラーウィンドウの下部にあるアクションメニュー、またはクラスグループのコンテキストメニューから 新規クラス... を選択します。



- エクスプローラーのホームページのコンテキストメニューより 新規 > クラス... を選択します。

クラスのコードサポート

各種 4D ウィンドウ (コードエディター、コンパイラ、デバッガー、ランタイムエクスプローラー) において、クラスコードは "特殊なプロジェクトメソッド" のように扱われます:

- コードエディター:
 - クラスは実行できません
 - クラスメソッドはコードのブロックです
 - オブジェクトメンバーに対する 定義に移動 操作はクラスの Function 宣言を探します。例: "\$o.f()" の場合、"Function f" を見つけます。
 - クラスのメソッド宣言に対する 参照箇所を検索 操作は、そのメソッドがオブジェクトメンバーとして使われている箇所を探します。例: "Function f" の場合 "\$o.f()" を見つけます。
- ランタイムエクスプローラーおよびデバッガーにおいて、クラスメソッドは <ClassName> コンストラクターまたは <ClassName>. <FunctionName> 形式で表示されます。

クラスストア

定義されたクラスには、クラスストアよりアクセスすることができます。クラスストアには次の二つが存在します:

- `cs` - ユーザークラスストア
- `4D` - ビルトインクラスストア

CS

`cs` -> classStore

引数	タイプ		説明
classStore	object	<-	プロジェクトまたはコンポーネントのユーザークラスストア

`cs` コマンドは、カレントプロジェクトまたはコンポーネントのユーザークラスストアを返します。これには、プロジェクトまたはコンポーネントにて 定義 されている、すべてのユーザークラスが含まれます。デフォルトでは、[ORDAクラス](#) のみ利用可能です。

例題

`myClass` オブジェクトの新規インスタンスを作成するには、次のように書きます:

```
$instance:=cs.myClass.new()
```

4D

`4D` -> classStore

引数	タイプ		説明
classStore	object	<-	4Dクラスストア

`4D` コマンドは、ビルトイン 4Dクラスのクラスストアを返します。[CryptoKey](#) などの専用 API へのアクセスを提供します。

例題

`CryptoKey` クラスに新規キーを作成するには、次のように書きます:

```
$key:=4D.CryptoKey.new(New object("type";"ECDSA";"curve";"prime256v1"))
```

Class オブジェクト

プロジェクトにおいてクラスが 定義 されていれば、それは 4Dランゲージ環境に読み込まれます。クラスとは、それ自身が "Class" クラス のオブジェクトです。Class オブジェクトは次のプロパティや関数を持ちます:

- `name` 文字列
- `superclass` オブジェクト (無い場合は null)
- `new()` 関数 (クラスオブジェクトをインスタンス化します)

また、Class オブジェクトは `constructor` オブジェクトを参照することも可能です。

Class オブジェクトは [共有オブジェクト](#) です。したがって、異なる 4Dプロセスから同時にアクセスすることができます。

継承

クラス宣言において `Class extends` キーワードを使うと、そのクラスは親クラス (つまり

スーパークラス) を継承します。

関数やプロパティがクラス内で見つからない場合、4D はそのクラスの [スーパークラス](#) 内を検索します。見つからない場合、4D はさらに、そのスーパークラスのスーパークラス内を探します。これは、スーパークラスが存在しなくなるまで続きます (すべてのオブジェクトは "Object" スーパークラスを継承してい

ます)。

クラスキーワード

クラス定義内では、専用の 4Dキーワードが使用できます:

- `Function <Name>` : オブジェクトのクラス関数を定義します。
- `Function get <Name>` と `Function set <Name>` : オブジェクトの計算プロパティを定義します。
- `Class Constructor` : オブジェクトのスタティックプロパティを定義します。
- `Class extends <ClassName>` : 繙承を定義します。

Function

シンタックス

```
Function <name>({$parameterName : type; ...}){->$parameterName : type}  
// コード
```

クラス関数とは、当該クラスのプロパティです。クラス関数は [4D.Function](#) クラスのオブジェクトです。

クラス定義ファイルでは、`Function` キーワードと関数名を使用して宣言をおこないます。関数名は [プロパティ名の命名規則](#) に準拠している必要があります。

Tip: アンダースコア ("_") 文字で関数名を開始すると、その関数は 4Dコードエディターの自動補完機能から除外されます。たとえば、`MyClass` に `Function _myPrivateFunction` を宣言した場合、コードエディターにおいて "cs.MyClass" とタイプしても、この関数は候補として提示されません。

関数名のすぐ後に、名前とデータ型を指定して [引数](#) を宣言します (戻り値の宣言も可)。たとえば:

```
Function computeArea($width : Integer; $height : Integer)->$area : Integer
```

クラスメソッド内でオブジェクトインスタンスを参照するには `This` コマンドを使います。たとえば:

```
Function setFullscreen($firstname : Text; $lastname : Text)  
  This.firstName:=$firstname  
  This.lastName:=$lastname  
  
Function getFullscreen()->$fullname : Text  
  $fullname:=This.firstName+" "+Uppercase(This.lastName)
```

クラスメソッドの場合には、`Current method name` コマンドは次を返します: `<ClassName>.<FunctionName>` (例: "MyClass.myFunction")。

アプリケーションのコード内では、クラス関数はオブジェクトインスタンスのメンバーメソッドとして呼び出され、[引数](#クラス関数の引数 mark=) を受け取ることができます。以下のシンタックスがサポートされています:

- () 演算子の使用。たとえば、`myObject.methodName("hello")`
- "4D.Function" クラスメンバーメソッドの使用: - `apply()`
 - `call()`

スレッドセーフに関する警告: クラス関数がスレッドセーフではないのに、"プリエンプティブプロセスで実行可能" なメソッドから呼び出された場合:
- 普通のメソッドの場合とは異なり、コンパイラーはエラーを生成しません。
- ランタイムにおいてのみ、4D はエラーを生成します。

関数の引数は、引数名とデータ型をコロンで区切って宣言します。パラメータ名は [プロパティ名の命名規則](#) に準拠している必要があります。複数のパラメーター（およびその型）を宣言する場合は、それらをセミコロン（;）で区切れます。

```
Function add($x; $y : Variant; $z : Integer; $xy : Object)
```

パラメーターの型が宣言されていない場合には、[バリアント](#) 型として定義されます。

メソッド内の引数宣言に使用される [従来の 4Dシンタックス](#) を、クラス関数の引数宣言に使うこともできます。両方のシンタックスは併用することができます。たとえば：

```
Function add($x : Integer)
  var $2; $value : Integer
  var $0 : Text
  $value:=$x+$2
  $0:=String($value)
```

戻り値

関数の戻り値を宣言するには（任意）、入力パラメータリストに矢印（->）と戻り値の定義を追加します。または、コロン（:）記号の後に戻り値のデータ型だけを指定することも可能です。たとえば：

```
Function add($x : Variant; $y : Integer)->$result : Integer
  $result:=$x+$y
```

コロン（:）記号の後に戻り値のデータ型だけを指定し、そのうえで [return 文](#) を使って戻り値を返すこともできます（これは関数の実行を終了します）。たとえば：

```
Function add($x : Variant; $y : Integer): Integer
  // なんらかのコード
  return $x+$y
```

例題 1

```
// クラス: Rectangle
Class constructor($width : Integer; $height : Integer)
  This.name:="Rectangle"
  This.height:=$height
  This.width:=$width

// 関数定義
Function getArea()->$result : Integer
  $result:=(This.height)*(This.width)
```

// プロジェクトメソッドにて

```
var $rect : cs.Rectangle
var $area : Real

$rect:=cs.Rectangle.new(50;100)
$area:=$rect.getArea() //5000
```

例題 2

`return` 文 を使った例です:

```
Function getRectArea($width : Integer; $height : Integer) : Integer
    If ($width > 0 && $height > 0)
        return $width * $height
    Else
        return 0
    End if
```

Function get と Function set

シンタックス

```
Function get <name>()->$result : type
// コード
```

```
Function set <name>($parameterName : type)
// コード
```

`Function get` と `Function set` は、クラスの 計算プロパティを定義するアクセサーです。計算プロパティとは、計算をマスクするデータ型を持つ命名プロパティです。計算プロパティの値にアクセスすると、4D は対応するアクセサーのコードを実行します:

- プロパティを読み取るときには `Function get` が実行されます。
- プロパティに書き込むときには `Function set` が実行されます。

プロパティがアクセスされない場合は、コードも実行されません。

計算プロパティは、メモリ上に保持する必要のないデータを処理するために設計されています。計算プロパティは通常、永続的なプロパティに基づいています。たとえば、クラスオブジェクトの永続的なプロパティとして、税込価格と消費税率が含まれている場合、税抜価格は計算プロパティで処理することができます。

クラス定義ファイルでは、計算プロパティの宣言には、`Function get` (ゲッター) と `Function set` (セッター) のキーワードを使い、その後にプロパティ名を記述します。名称は [プロパティ名の命名規則](#) に準拠している必要があります。

`Function get` はプロパティの型の値を返し、`Function set` はプロパティの型の引数を受け取ります。どちらも、標準的な [関数の引数](#) のルールに準拠する必要があります。

両方の関数が定義されている場合、計算プロパティは read-write となります。`Function get` のみが定義されている場合、計算プロパティは read-only です。この場合、コードがプロパティを変更しようとするとエラーが返されます。`Function set` のみが定義されている場合、4D はプロパティの読み取り時に `undefined` を返します。

計算プロパティの型は、ゲッターの `$return` の型宣言によって定義されます。[有効なプロパティタイプ](#) であれば、いずれも使用可能です。

オブジェクトプロパティに `undefined` を代入すると、型を保持したまま値がクリアされます。このためには、まず `Function get` を呼び出して値の型を取得し、次にその型の空の値で `Function set` を呼び出します。

例題 1

```
// クラス: Person.4dm

Class constructor($firstname : Text; $lastname : Text)
    This.firstName:=$firstname
    This.lastName:=$lastname

Function get fullName() -> $fullName : Text
    $fullName:=This.firstName+" "+This.lastName

Function set fullName( $fullName : Text )
    $p:=Position(" "; $fullName)
    This.firstName:=Substring($fullName; 1; $p-1)
    This.lastName:=Substring($fullName; $p+1)
```

```
// プロジェクトメソッドにて
$fullName:=$person.fullName // Function get fullName() が呼び出されます
$person.fullName:="John Smith" // Function set fullName() が呼び出されます
```

例題 2

```
Function get fullAddress()->$result : Object

$result:=New object

$result.fullName:=This.fullName
$result.address:=This.address
$result.zipCode:=This.zipCode
$result.city:=This.city
$result.state:=This.state
$result.country:=This.country
```

Class Constructor

シンタックス

```
// クラス: MyClass
Class Constructor({$parameterName : type; ...})
// コード
```

クラスコンストラクター関数を使って、ユーザークラスを定義することができます。このコンストラクターは [引数](#) を受け取ることができます。

クラスコンストラクター関数の場合には、 `Current method name` コマンドは次を返します: `<ClassName>:constructor` (例: "MyClass:constructor")。

クラスコンストラクター関数の場合には、 `Current method name` コマンドは次を返します: `<ClassName>:constructor` (例: "MyClass:constructor")。

例:

```
// クラス: MyClass
// MyClass のクラスコンストラクター
Class Constructor ($name : Text)
    This.name:=$name
```

```
// プロジェクトメソッドにて  
// オブジェクトをインスタンス化します  
var $o : cs.MyClass  
$o:=cs.MyClass.new("HelloWorld")  
// $o = {"name":"HelloWorld"}
```

Class extends <ClassName>

シンタックス

```
// クラス: ChildClass  
Class extends <ParentClass>
```

クラス宣言において `Class extends` キーワードを使うと、別のユーザークラスの子ユーザークラスを作成することができます。この子クラスは、親クラスのすべての機能を継承します。

クラス継承は次のルールに沿っている必要があります:

- ユーザークラスはビルトインクラスを継承できません (例外は `4D.Object` で、すべてのユーザークラスにデフォルトで継承されます)
- ユーザークラスは、別のプロジェクトやコンポーネントのユーザークラスを継承できません。
- ユーザークラスは、自身を継承することはできません。
- 間接的にも、自身を継承することはできません (例: "a" extends "b" かつ "b" extends "a")。

クラス継承は次のルールに沿っている必要があります:

派生クラスは、`Super` コマンドを使って親クラスのコンストラクターを呼び出すことができます。

例題

派生クラスは、`Super` コマンドを使って親クラスのコンストラクターを呼び出すことができます。

```
//クラス: Square  
  
// パス: Classes/Square.4dm  
  
Class extends Polygon  
  
Class constructor ($side : Integer)  
  
    // 親クラスのコンストラクターを呼び出します  
    // 長方形の高さ・幅パラメーターに正方形の一辺の長さを引数として渡します  
    Super($side;$side)  
    // 派生クラスにおいては、'This' を使用するより先に  
    // Super を呼び出しておく必要があります  
    This.name:="Square"  
  
Function getArea()  
    C_LONGINT($0)  
    $0:=This.height*This.width
```

Super

シンタックス

```
Super {{ param{;...;paramN} }} {-> Object}
```

引数	タイプ		説明
param	混合	->	親コンストラクターに受け渡す引数
戻り値	object	<-	親オブジェクト

`Polygon` クラスを継承した `Square` クラスを作成します。

`Super` キーワードによってスーパークラス（親クラス）を呼び出すことができます。

1. `コンストラクタコード` 内において、`Super` はスーパークラスのコンストラクターを呼び出すコマンドです。コンストラクター内で使用する際に `Super` コマンドは単独で使用され、また `This` キーワードよりも先に使用される必要があります。
 - 継承ツリーにおいて、すべてのクラスコンストラクターが正しく呼び出されていない場合には、エラー -10748 が生成されます。呼び出しが有効であることを確認するのは、開発者の役目となります。
 - スーパークラスがコンストラクトされるより前に、`This` コマンドを使った場合には、エラー -10743 が生成されます。
 - オブジェクトのスコープ外で `Super` を呼び出した場合、または、スーパークラスコンストラクターがすでに呼び出されたオブジェクトを対象に呼び出した場合には、エラー -10746 が生成されます。

```
// myClass コンストラクター
var $text1; $text2 : Text
Super($text1) // テキスト型引数をスーパークラスコンストラクターに渡します
This.param:=$text2 // 2番目の引数を使用します
```

2. `クラスメンバー関数` 内において、`Super` はスーパークラスのプロトタイプを指し、スーパークラス階層のメンバーメソッドの呼び出しを可能にします。

```
Super.doSomething(42) // スーパークラスにて宣言されている
// "doSomething" メンバーメソッドを呼び出します
```

例題 1

クラスコンストレクター内で `Super` を使う例です。`Rectangle` と `Square` クラス の共通要素がコンストラクター内で重複しないよう、このコマンドを呼び出します。

```
// クラス: Rectangle
Class constructor($width : Integer; $height : Integer)
  This.name:="Rectangle"
  This.height:=$height
  This.width:=$width

Function sayName()
  ALERT("Hi, I am a "+This.name+".")

// 関数定義
Function getArea()
  var $0 : Integer
  $0:=(This.height)*(This.width)
```

```
// クラス: Square

Class extends Rectangle

Class constructor ($side : Integer)

    // 親クラスのコンストラクターを呼び出します
    // 長方形の高さ・幅パラメーターに正方形の一辺の長さを引数として渡します
    Super($side;$side)
    // 派生クラスにおいては、'This' を使用するより先に
    // Super を呼び出しておく必要があります
    This.name:="Square"

Function getArea()
    C_LONGINT($0)
    $0:=This.height*This.width
```

例題 2

クラスメンバー関数内で `Super` を使う例です。メンバー関数を持つ `Rectangle` クラスを作成します:

```
// クラス: Rectangle

Function nbSides()
    var $0 : Text
    $0:="I have 4 sides"
```

`Square` クラスには、スーパークラス関数を呼び出すメンバー関数を定義します:

```
// クラス: Square

Class extends Rectangle

Function description()
    var $0 : Text
    $0:=Super.nbSides()+" which are all equal"
```

`Square` クラスには、スーパークラス関数を呼び出すメンバー関数を定義します:

```
var $square : Object
var $message : Text
$square:=cs.Square.new()
$message:=$square.description() // I have 4 sides which are all equal
```

This

シンタックス

`This` → Object

引数	タイプ		説明
戻り値	object	<-	カレントオブジェクト

`This` キーワードは、現在処理中のオブジェクトへの参照を返します。`This` は、4Dにおいて [様々なコンテキスト](#) で使用することができます。

`This` の値は、呼び方によって決まります。`This` の値は実行時に代入により設定することはできません。また、呼び出されるたびに違う値となります。

オブジェクトのメンバーメソッドとしてフォーミュラが呼び出された場合、`This` はメソッドの呼び出し元であるオブジェクトを指します。たとえば：

```
$o:=New object("prop";42;"f";Formula(This.prop))
$val:=$o.f() //42
```

クラスコンストラクター 関数が `new()` 関数により使用された場合、その内部の `This` はインスタンス化される新規オブジェクトを指します。

```
// クラス: ob

Class Constructor

// This のプロパティを
// 代入によって作成します
This.a:=42
```

```
// 4Dメソッドにて
$o:=cs.ob.new()
$val:=$o.a //42
```

コンストラクター内で `Super` キーワードを使ってスーパークラスのコンストラクターを呼び出す場合、必ず `This` より先にスーパークラスのコンストラクターを呼ぶ必要があることに留意してください。こちらの [例題](#) を参照ください。

クラスコンストラクター 関数が `new()` 関数により使用された場合、その内部の `This` はインスタンス化される新規オブジェクトを指します。

```
// クラス: ob

Function f()
$0:=This.a+This.b
```

`Square` クラスには、スーパークラスメソッドを呼び出すメンバーメソッドを定義します：

```
$o:=cs.ob.new()
$o.a:=5
$o.b:=3
$val:=$o.f() //8
```

この例では、変数 `$o` に代入されたオブジェクトは `f` プロパティを持たないため、これをクラスより継承します。`f` は `$o` のメソッドとして呼び出されるため、メソッド内の `This` は `$o` を指します。

クラスコマンド

4Dランゲージには、クラス機能を扱う複数のコマンドがあります。

OB Class

`OB Class (object) -> Object | Null`

4Dランゲージには、クラス機能を扱う複数のコマンドがあります。

OB Instance of

OB Instance of (object ; class) -> Boolean

OB Class は引数として渡したオブジェクトのクラスを返します。

制御フロー

メソッドや関数が単純か複雑かに関係なく、開発者は3つのプログラミング構造のうち、1つ以上を常に使用します。プログラミング構造は、メソッド内でステートメントが実行される順序を決定する実行フローをコントロールします。3つのタイプの構造があります：

- **シーケンシャル**: シーケンシャル構造は単純な線形構造です。シーケンスとは、4Dが最初から最後まで次々に実行する一連のステートメントです。オブジェクトメソッドで頻繁に使用される1行から成るルーチンはもっとも簡単なシーケンシャル構造の例です。例：
`[People] lastName:=Uppercase ([People] lastName)`
- **分岐**: 分岐構造は、条件をテストし、その結果に基づいて異なる流れにメソッドを導きます。条件は `true` または `false` に評価される布尔式です。`If...Else...End if` 構文は分岐構造の一例で、処理フローを二つに分岐します。`Case of...Else...End case` 構文も分岐構造の一つで、処理フローをもとたくさん分岐することができます。
- **ループ**: メソッドの作成にあたって、何度も同じ処理を繰り返すことがあります。これに実現するために、4Dは以下のループ構造を備えています：
 - `While...End while`
 - `Repeat...Until`
 - `For...End for`
 - `For each...End for each`

ループを制御する方法には、条件が満たされるまでループする方法と、指定した回数だけループする方法の2通りがあります。各ループ構造はいずれの方法にも用いることができますが、`While` ループと `Repeat` ループは条件が満たされるまで繰り返す場合に、`For` ループは指定した回数だけループする場合の利用に適切です。`For each...End for each` ループは両方を組み合わせることが可能で、オブジェクトやコレクション内でループするために設計されています。

注：4Dはプログラム構造 (`If/While/For/Caes of/Repeat/For each`) を512レベルまで入れ子で記述できます。

return {expression}

▶ 補足

`return` 文はどこからでも呼び出すことができます。関数やメソッドの中で `return` 文が使われると、その関数やメソッドの実行が中断されます。残りのコードは実行されず、呼び出し元に制御が返されます。

`return` 文を使用して、呼び出し元に戻り値を返すことができます。

例題

```
var $message : Text
var $i : Integer

While (True) // 無限ループ
    $i:=$i+1
    $message+=String($i)+"A\r" // 5まで実行されます
    logConsole($message)
    If ($i=5)
        return // ループを終了させます
    End if
    $message+=String($i)+"B\r" // 4まで実行されます
    logConsole($message)
End while
$message+=String($i)+"C\r" // 実行されることはありません
logConsole($message)

// 1A
// 1B
// 2A
// 2B
// 3A
// 3B
// 4A
// 4B
// 5A
```

分岐構造

分岐構造は、条件をテストし、その結果に基づいて異なる流れにメソッドを導きます。

If...Else...End if

If...Else...End if による制御フロー構造の正式な構文は以下のようになります:

```
If(Boolean_Expression)
    statement(s)
Else
    statement(s)
End if
```

Else 部分はオプションであり、省略して以下のように記述できます:

```
If(Boolean_Expression)
    statement(s)
End if
```

If...Else...End if 構造は、条件 (ブール式) が true か false かによって、処理の選択肢を2つメソッドに与えます。ブール式が true の場合は、テストのすぐ後のステートメントを実行し、ブール式が FALSE の場合には、Else 文のすぐ後のステートメントを実行します。任意の Else が省略されていた場合、End if のすぐ後のステートメント (あれば) へと実行が続行されます。

ブール式は常に全体が評価されるという点に注意してください。たとえば、以下のような場合:

```
If(MethodA & MethodB)
    ...
End if
```

この場合、両方のメソッドが true である場合に限り、式は true になります。しかしながら MethodA が false であっても、4DはMethodB も評価するため、これは時間の無駄になります。この場合には、以下のような構造を使用するほうが賢明といえます:

```
If(MethodA)
    If(MethodB)
        ...
    End if
End if
```

上記の結果はほぼ同じで、MethodB は必要な場合にのみ評価されます。

注記: 三項演算子 を使うことで、条件式を 1行で書くことができ、If...Else 文を置き換えることもできます。

例題

```
// ユーザーに名前の入力を求めます  
$Find:=Request("名前を入力してください")  
If(OK=1)  
    QUERY([People];[People]LastName=$Find)  
Else  
    ALERT("名前が入力されませんでした")  
End if
```

Tip: 一方の条件に実行ステートメントがない分岐処理を書くこともできます。下のようなコードはどちらも有効です:

```
If(Boolean_Expression)  
Else  
    statement(s)  
End if
```

または:

```
If(Boolean_Expression)  
    statement(s)  
Else  
End if
```

Case of...Else...End case

Case of...Else...End case による制御フロー構造の正式な構文は以下のようになります:

```
Case of  
:(Boolean_Expression)  
    statement(s)  
:(Boolean_Expression)  
    statement(s)  
:  
:  
:  
:(Boolean_Expression)  
    statement(s)  
Else  
    statement(s)  
End case
```

Else 部分はオプションであり、省略して以下のように記述できます:

```
Case of  
:(Boolean_Expression)  
    statement(s)  
:(Boolean_Expression)  
    statement(s)  
:  
:  
:(Boolean_Expression)  
    statement(s)  
End case
```

`If...Else...End if` と同様に、`Case of...Else...End case` 構造も処理の選択肢をメソッドに与えます。`If...Else...End` との違いは、`Case of...Else...End case` 構造が複数のブール式を評価し、その中から最初に true となるステートメントを実行することです。

ブール式の前にはそれぞれコロン（：）を付けます。コロンとブール式の組み合わせをケースと呼びます。例えば以下の行はケースです：

```
: (bValidate=1)
```

最初に true になったケースに続く（次のケースまでの）ステートメントだけが実行されます。true になるケースがない場合、どのステートメントも実行されません（`Else` 文が指定されていない場合）。

最後のケースの後に `Else` 文を含むことができます。すべてのケースが FALSE の場合に、`Else` 文の後のステートメントが実行されます。

例題

下記の例は数値変数を判定し、対応する数字をアラートボックスに表示します：

```
Case of
  :(vResult=1) // 数値が1の場合
    ALERT("一です。") // 1のアラートボックスを表示します
  :(vResult=2) // 数値が2の場合
    ALERT("二です。") // 2のアラートボックスを表示します
  :(vResult=3) // 数値が3の場合
    ALERT("三です。") // 3のアラートボックスを表示します
 Else // 数値が1,2,3のいずれでもない場合
   ALERT("一、二、三のいずれでもありません。")
End case
```

比較するために、同じことを `If...Else...End if` 構文で記述すると以下のようにになります。

```
If(vResult=1) // 数値が1の場合
  ALERT("一です。") // 1のアラートボックスを表示します
Else
  If(vResult=2) // 数値が2の場合
    ALERT("二です。") // 2のアラートボックスを表示します
  Else
    If(vResult=3) // 数値が3の場合
      ALERT("三です。") // 3のアラートボックスを表示します
    Else // 数値が1,2,3のいずれでもない場合
      ALERT("一、二、三のいずれでもありません。")
    End if
  End if
End if
```

`Case of...Else...End case` 構造は、最初に true になったケースだけを実行します。2つ以上のケースが true の場合は、最初に true になったケースのステートメントだけを実行します。

したがって、階層的なテストを実行するときには、階層上で低い位置にある条件がテスト順序で先に記述されていることを確認する必要があります。以下の例では、ケース2が true の場合、ケース1も必ず true であるため、ケース1は後に位置すべきです。このままの順序では、ケース2のステートメントはけっして実行されません：

```
Case of
  :(vResult=1)
  ...
// ステートメントなど
  :((vResult=1) & (vCondition#2)) // このケースが判定されることはありません
  ...
// ステートメントなど
End case
```

vResult = 1の判定により他の条件を見る前に分岐するので、第2のケースが判定されることはありません。コードが正しく実行されるためには次のように書きます：

```
Case of
  :((vResult=1) & (vCondition#2)) // このケースが先に判定されます
  ...
  // ステートメントなど
  :(vResult=1)
  ...

  // ステートメントなど
End case
```

さらに階層的なテストを実行したい場合、コードも階層化する必要があります。

Tip: 分岐構造において、ケースに続くステートメントの記述は必須ではありません。下のようなコードはどちらも有効です：

```
Case of
  :(Boolean_Expression)
  :(Boolean_Expression)
  ...
  :(Boolean_Expression)
    statement(s)
  Else
    statement(s)
End case
```

または：

```
Case of
  :(Boolean_Expression)
  :(Boolean_Expression)
    statement(s)
  ...
  :(Boolean_Expression)
    statement(s)
  Else
End case
```

または：

```
Case of
  Else
    statement(s)
End case
```

ループ構造

ループ構造は、条件が満たされるまで、あるいは指定した回数まで、同じ処理を繰り返します。

While...End while

While...End while による制御フロー構造の正式な構文は以下のようになります:

```
While(Boolean_Expression)
  statement(s)
  {break}
  {continue}
End while
```

While...End while ループは、布尔式が true である限り、ループ内のステートメントを実行し続けます。ループの始めに布尔式を評価し、布尔式が FALSE の場合にはループをおこないません。

break および continue ステートメントについては [後述します](#)。

一般に、While...End while ループに入る手前で、布尔式で判定する値を初期化しておきます。通常は布尔式が true になるように設定してからループに入ります。

布尔式は、ループ内の要素を使って設定されなければなりません。そうでなければ、ループは永久に続くでしょう。以下の例では、NeverStop がいつも true であるので、ループは永久に続きます。

```
NeverStop:=True
While(NeverStop)
End while
```

このようにメソッドの実行が制御不能になった場合には、トレース機能を使用し、ループを止めて、問題点を追跡することができます。メソッドのトレース方法については、[エラー処理](#) の章を見てください。

例題

```
CONFIRM("新規レコードを追加しますか?") // ユーザーに確認します
While(OK=1) // 利用者が望む限りループします
  ADD RECORD([aTable]) // 新規にレコードを追加します
End while // ループは必ず End while によって終わります
```

この例では、まずループに入る前に CONFIRM コマンドによりシステム変数 OK がセットされます。ユーザーがダイアログボックスで OK ボタンをクリックすると、システム変数 OK に1がセットされ、ループを開始します。それ以外の場合はシステム変数 OK に0が設定され、ループをスキップします。ループに入ると、ADD RECORD コマンドはループを続けます。これは、ユーザーがレコードを保存した時点で、システム変数 OK に1が設定されるからです。ユーザーが最後のレコードを取り消した（保存しない）時点で、システム変数 OK に0がセットされ、ループは終了します。

Repeat...Until

Repeat...Until による制御フロー構造の正式な構文は以下のようになります:

```
Repeat
  statement(s)
  {break}
  {continue}
Until(Boolean_Expression)
```

`Repeat...Until` ループは、`While...End while` ループと似ていますが、まずループの後で布尔式を判定する点が異なります。つまり、`Repeat...Until` ループは最低でも1回は必ずループを実行しますが、`While...End while` ループは最初の布尔式が FALSE である場合には、ループを1回も実行しません。

もう一つの `While...End while` ループとの相違点は、`Repeat...Until` は布尔式が true になるまでループを続行することです。

`break` および `continue` ステートメントについては [後述します](#)。

例題

以下の例を、`While...End while` ループの例と比較してください。布尔式を、初期化しておく必要がない点に注目してください。システム変数 `OK` を初期化する `CONFIRM` コマンドはありません。

```
Repeat
  ADD RECORD( [aTable])
Until(OK=0)
```

For...End for

`For...End for` による制御フロー構造の正式な構文は以下のようになります:

```
For(Counter_Variable;Start_Expression;End_Expression{;Increment_Expression})
  statement(s)
  {break}
  {continue}
End for
```

`For...End for` ループは、カウンター変数によりループを制御します:

- カウンター変数 `Counter_Variable` は、数値変数（実数または倍長整数）で、`Start_Expression` に指定した値に初期化されます。
- ループを実行するたびに、任意の引数である `Increment_Expression` の値がカウンター変数に加算されます。`Increment_Expression` を指定しない場合、増分値は1になります。
- カウンターが `End_Expression` の値を超えた時点で、ループを停止します。

重要: `Start_Expression`、`End_Expression`、`Increment_Expression` の値は、ループのために一度だけ評価されます。これらの数値が変数で指定されている場合、ループ内でその変数の値を変更してもループは影響を受けません。

Tip: 特別な目的のために、カウンター変数 `Counter_Variable` の値を変更することができます。ループ内でカウンター変数を変更すると、ループはその影響を受けます。

- 通常、`Start_Expression` は `End_Expression` より小さい。
- `Start_Expression` と `End_Expression` が等しい場合、1回だけループがおこなわれます。
- `Start_Expression` が `End_Expression` より大きい場合、`Increment_Expression` に負の値を指定しない限り、ループはおこなわれません。次に例を示します。

`break` および `continue` ステートメントについては [後述します](#)。

基本的な使用例

- 以下の例は、100回の繰り返しをおこないます:

```
For(vCounter;1;100)
// なんらかの処理
End for
```

2. 以下の例は、配列 anArray の全要素に対して処理をおこないます：

```
For($vlElem;1;Size of array(anArray))
// 各配列要素に対する処理
anArray{$vlElem}:=...
End for
```

3. テキスト変数 vtSomeText の文字を一つ一つループ処理します：

```
For($vlChar;1;Length(vtSomeText))
// 文字がタブであれば
If(Character code(vtSomeText[$vlChar])=Tab)
// なんらかの処理をします
End if
End for
```

4. 以下の例は、テーブル [aTable] のカレントセクションの各レコードについて処理をおこないます：

```
FIRST RECORD([aTable])
For($vlRecord;1;Records in selection([aTable]))
// 各レコードに対する処理
SEND RECORD([aTable])
////
// 次レコードへ移動します
NEXT RECORD([aTable])
End for
```

プロジェクトで作成する大部分の `For...End for` ループは、上記例題のいずれかの形式になるでしょう。

カウンター変数の減算

ループに際してカウンター変数を増加させるのではなく、減少させたい場合があります。その場合、*Start_Expression* に *End_Expression* より大きい値を設定し、*Increment_Expression* に負の数を指定する必要があります。次に挙げる例題は、前述の例と同じ処理を逆の順序でおこないます：

5. 以下の例は、100回の繰り返しをおこないます：

```
For(vCounter;100;1;-1)
// なんらかの処理
End for
```

6. 以下の例は、配列 anArray の全要素に対して処理をおこないます：

```
For($vlElem;Size of array(anArray);1;-1)
// 各配列要素に対する処理
anArray{$vlElem}:=...
End for
```

7. テキスト変数 vtSomeText の文字を一つ一つループ処理します：

```

For($vlChar;Length(vtSomeText);1;-1)
    // 文字がタブであれば
    If(Character_code(vtSomeText[$vlChar])=Tab)
        // なんらかの処理をします
    End if
End for

```

8. 以下の例は、テーブル [aTable] のカレントセクションの各レコードについて処理をおこないます:

```

FIRST RECORD([aTable])
For($vlRecord;Records in selection([aTable]);1;-1)
    // 各レコードに対する処理
    SEND RECORD([aTable])
    //...
    // 前レコードへ移動します
    PREVIOUS RECORD([aTable])
End for

```

1より大きな値によるカウンター変数の増加

必要に応じて、*Increment_Expression* (正または負の値) に、その絶対値が1より大きな値を指定できます。

9. 以下の例は、配列 anArray の偶数要素について処理を行います:

```

For($vlElem;2;Size of array(anArray);2)
    // 偶数要素 #2,#4...#2n に対する処理
    anArray{$vlElem}:=...
End for

```

ループ構造の比較

`For...End for` ループの例をもう一度見てみましょう。以下の例は、100回の繰り返しをおこないます:

```

For(vCounter;1;100)
    // なんらかの処理
End for

```

`While...End while` ループと `Repeat...Until` ループで、同じ処理を実行する方法を調べてみましょう。以下は、同じ処理を実行する `While...End while` ループです:

```

$i:=1 // カウンターの初期化
While($i<=100) // 100回のループ
    // なんらかの処理
    $i:=$i+1 // カウンターの増分が必要
End while

```

同じことを `Repeat...Until` ループで記述すると以下のようになります:

```

$i:=1 // カウンターの初期化
Repeat
    // なんらかの処理
    $i:=$i+1 // カウンターの増分が必要
Until($i=100) // 100回のループ

```

Tip: `For...End for` ループは、`While...End while` や `Repeat...Until` ループよりも高速です。これは4Dが内部的にカウンター変数のテストおよび増加を行うからです。したがって、可能な限り `For...End for` ループの使用が推奨されます。

For...End for ループの最適化

カウンター変数（インタープロセス、プロセス、ローカル変数）には実数、または倍長整数タイプを使用します。数多く繰り返されるループの場合、とくにコンパイルモードでは、倍長整数タイプのローカル変数を使用してください。

10. 次に例を示します：

```
C_LONGINT($vlCounter) // 倍長整数型のローカル変数を使用します
For($vlCounter;1;10000)
// なんらかの処理
End for
```

For...End for の入れ子構造

制御構造は、必要に応じて入れ子にする（ネストする）ことができます。`For...End for` ループも同じです。誤りを避けるため、各ループ構造ごとに別のカウンター変数を使用してください。

次に例を示します：

1. 以下の例は二次元配列の全要素への処理です：

```
For($vlElem;1;Size of array(anArray))
////
// 各行に対する処理
////
For($vlSubElem;1;Size of array(anArray{$vlElem}))
// 各要素に対する処理
    anArray{$vlElem}{$vlSubElem}:=...
End for
End for
```

2. 以下の例は、データベースのすべての日付フィールドに対するポインターの配列を作成します：

```
ARRAY POINTER($apDateFields;0)
$vlElem:=0
For($vlTable;1;Get last table number)
    If(Is table number valid($vlTable))
        For($vlField;1;Get last field number($vlTable))
            If(Is field number valid($vlTable;$vlField))
                $vpField:=Field($vlTable;$vlField)
                If(Type($vpField->)=Is date)
                    $vlElem:=$vlElem+1
                    INSERT IN ARRAY($apDateFields;$vlElem)
                    $apDateFields{$vlElem}:=$vpField
                End if
            End if
        End for
    End if
End for
```

For each...End for each

`For each...End for each` による制御フロー構造の正式な構文は以下のようになります：

```

For each(Current_Item;Expression{;begin{}};end{}){Until|While}(Boolean_Expression)
  statement(s)
  {break}
  {continue}
End for each

```

`For each...End for each` 構造は、*Expression* に含まれるすべての *Current_item* に対して処理を繰り返します。*Current_item* の型は *Expression* の型に依存します。`For each...End for each` ループは3種類の **Expression* * を対象に反復処理をおこなうことができます：

- コレクション：コレクションの各要素をループします
- エンティティセレクション：各エンティティをループします
- オブジェクト：各オブジェクトプロパティをループします

以下の表は、`For each...End for each` の3つのタイプを比較したものです：

	コレクション内のループ	エンティティセレクション内のループ	オブジェクト内のループ
Current_Item の型	コレクション要素と同じ型の変数	エンティティ	テキスト変数
Expression の型	(同じ型の要素を持つ) コレクション	エンティティセレクション	オブジェクト
ループ数 (デフォルト)	コレクションの要素数	セレクション内のエンティティ数	オブジェクトのプロパティ数
begin / end パラメーターをサポート	○	○	×

- ループの数は開始時に評価され、処理中に変化することはありません。ループ中に項目を追加・削除することは、繰り返しの不足・重複を引き起こすことがあるため、一般的には推奨されません。
- デフォルトでは、内部の *statement(s)* 部の処理は、*Expression* の各項目に対して実行されます。しかしながら、ループの先頭 (`While`) あるいはループの終わり (`Until`) で条件をテストすることで、ループを抜け出すことは可能です。
- 任意の *begin* および *end* パラメーターを指定することで、コレクションおよびエンティティセレクションに対してループの範囲を定義することができます。
- `For each...End for each` ループは 共有コレクション や 共有オブジェクト に対して使用することもできます。コレクションの要素またはオブジェクトのプロパティを変更する場合は、`Use...End use` 構文も追加が必要です。`Use...End use` 構文の使い方は、つぎのように状況に応じて異なります：
 - 整合性のため要素やプロパティを一括で処理しなくてはならない場合には、ループに入る前 (外側) に使います。
 - 要素やプロパティを個々に変更して差し支えない場合は、ループの中で使います。

`break` および `continue` ステートメントについては [後述します](#)。

コレクション内のループ

`For each...End for each` が *Collection* 型の *Expression* に対して使用された場合、*Current_Item* はコレクション要素と同じ型の変数です。デフォルトでは、ループの回数はコレクションの要素数に基づいています。

コレクションの要素はすべて同じ型でなくてはなりません。そうでない場合には、*Current_Item* 変数に別の型の値が代入されたときにエラーが生成されます。

各ループの繰り返しにおいて、*Current_Item* 変数には、合致するコレクションの要素が自動的に代入されます。このとき、以下の点に注意する必要があります：

- Current_Item* 変数がオブジェクト型あるいはコレクション型であった場合 (つまり *Expression* がオブジェクトのコレクション、あるいはコレクションのコレクションであった場合)、この変数を変更すると自動的にコレクションの対応する要素も変更されます (オブジェクトとコレクションは同じ参照を共有しているからです)。変数がスカラー型である場合、変数のみが変更されます。
- Current_Item* 変数は、コレクション要素の型と合致している必要があります。コレクション要素のどれか一つでも、変数と異なる型のものがあった場合、エラーが生成され、ループは停止します。
- コレクションが Null 値の要素を格納していたとき、*Current_Item* 変数の型が Null 値をサポートしない型 (倍長整数変数など) であった場合にはエラーが生成されます。

例題

数値のコレクションを対象に、統計情報を計算します：

```

C_COLLECTION($nums)
$nums:=New collection(10;5001;6665;33;1;42;7850)
C_LONGINT($item;$vEven;$vOdd;$vUnder;$vOver)
For each($item;$nums)
  If($item%2=0)
    $vEven:=$vEven+1
  Else
    $vOdd:=$vOdd+1
  End if
  Case of
    :($item<5000)
      $vUnder:=$vUnder+1
    :($item>6000)
      $vOver:=$vOver+1
  End case
End for each
//$vEven=3, $vOdd=4
//$vUnder=4, $vOver=2

```

エンティティセレクション内のループ

`For each...End for each` が *Entity selection* 型の *Expression* に対して使用された場合、*Current_Item* は現在処理中のエンティティです。

ループの回数はエンティティセレクション内のエンティティの数に基づきます。各ループの繰り返しにおいて、*Current_Item* には、処理の対象であるエンティティセレクション内のエンティティが自動的に代入されます。

注: エンティティセレクション内のエンティティが、途中で他のプロセスによって削除された場合、そのエンティティはループにおいて自動的にスキップされます。

カレントエンティティに対して適用された変更は、`entity.save()` で明示的に保存する必要があることに注意してください。

例題

`Employees` データクラスの中から、英国の従業員の給与を引き上げます:

```

C_OBJECT(emp)
For each(emp;ds.Employees.query("country='UK'"))
  emp.salary:=emp.salary*1,03
  emp.save()
End for each

```

オブジェクト内のループ

`For each...End for each` が *Object* 型の *Expression* に対して使用された場合、*Current_Item* は現在処理中のプロパティ名が自動代入されたテキスト変数です。

オブジェクトのプロパティは作成順に処理されています。ループ中、プロパティをオブジェクトに追加/削除することが可能ですが、その場合でも残りのループ回数は、オブジェクトの元のプロパティ数に基づいていますため、変化しません。

例題

下のオブジェクトに格納されている名前に関するプロパティの値をすべて大文字に変えます:

```
{
  "firstname": "gregory",
  "lastname": "badikora",
  "age": 20
}
```

以下のように書くことができます:

```
For each(property;vObject)
    If Value type(vObject[property])=Is text)
        vObject[property]:=Uppercase(vObject[property])
    End if
End for each
```

```
{
    "firstname": "GREGORY",
    "lastname": "BADIKORA",
    "age": 20
}
```

begin / end パラメーター

任意の begin と end パラメーターを指定することで、繰り返しの範囲を定義することができます。

注: begin と end パラメーターは、コレクションおよびエンティティセレクション型に対するループにおいてのみ使用することができます（オブジェクト型のときは無視されます）。

- begin には、Expression においてループを開始したい要素位置を渡します（このとき begin の値が指す要素はループに含まれます）。
- end には、Expression においてループを終了する要素位置を渡します（このとき end の値が指す要素はループに含まれません）。

end が省略されている、あるいは end が Expression の要素数より大きい場合、begin 引数の位置から最後の要素まで（含まれる）をループします。begin と end が正の値の場合、それらは Expression 内の要素の実際の位置を表します。begin 引数が負の値の場合、それは begin:=begin+Expression のサイズ として再計算されます（つまり、Expression の終端からのオフセットであるとみなされます）。再計算された値も負の値だった場合、begin は 0 に設定されます。注: begin が負の値だったとしても、繰り返しそのものは標準の順番で実行されます。end が負の値だった場合、それは end:=end+Expression のサイズ として再計算されます。

たとえば:

- コレクションには 10 の要素が格納されています（ナンバリングは #0 から #9）
- begin=-4 -> begin=-4+10=6 -> ループは 6 番目の要素 (#5) から開始されます
- end=-2 -> end=-2+10=8 -> 繰り返しは 8 番目の要素 (#7) の前に終了します、つまり 7 番目 (#6) の要素の処理が最後のループとなります。

例題

```
C_COLLECTION($col;$col2)
$col:=New collection("a";"b";"c";"d";"e")
$col2:=New collection(1;2;3)
C_TEXT($item)
For each($item;$col;0;3)
    $col2.push($item)
End for each
// $col2=[1,2,3,"a","b","c"]
For each($item;$col;-2;-1)
    $col2.push($item)
End for each
// $col2=[1,2,3,"a","b","c","d"]
```

Until と While 条件

For each...End for each の実行は、Until あるいは While 条件を追加することでコントロールすることができます。

Until(condition) 条件がループに組み込まれた場合、condition の式が true に評価されるとループは停止します。While(condition) 条件の場合は逆に、condition の式が false になるとループが停止します。

使用する条件は状況に応じて選べます:

- `Until` 条件は各ループの終わりにテストされます。そのため、*Expression* が空あるいは `null` でないかぎり、ループは少なくとも1回は実行されます。
- `While` 条件は各ループの始めにテストされます。そのため、評価の結果次第では、ループは一度も実行されないこともあります。

例題

```
$colNum:=New collection(1;2;3;4;5;6;7;8;9;10)

$total:=0
For each($num;$colNum)While($total<30) // 最初にテストされます
    $total:=$total+$num
End for each
ALERT(String($total)) // $total = 36 (1+2+3+4+5+6+7+8)

$total:=1000
For each($num;$colNum)Until($total>30) // 最後にテストされます
    $total:=$total+$num
End for each
ALERT(String($total)) // $total = 1001 (1000+1)
```

break と continue

上記のループ構造はすべて、`break` 文および `continue` 文をサポートしています。これらの文は、ループを完全に終了させたり、現在の繰り返しだけを終了させたりすることで、ループをよりコントロールすることができます。

break

`break` 文は、その文が含まれるループを終了させます。プログラムの制御は、ループ直後のステートメントに移ります。

[入れ子になったループ](#) (ループ内に別のループがある) の中に `break` 文がある場合、`break` 文は最も内側のループを終了させます。

例題

```
For (vCounter;1;100)
    If ($tab{vCounter}=="") // 条件が true になった場合
        break // forループを終了させます
    End if
End for
```

continue

`continue` 文は、ループにおいて現在実行中の繰り返しだけを終了させ、次の繰り返しよりループの実行を継続させます。

```
var $text : Text
For ($i; 0; 9)
    If ($i=3)
        continue // 次の繰り返しに移行します
    End if
    $text:=$text+String($i)
End for
// $text="012456789"
```

エラー処理

エラー処理とは、アプリケーション内で発生する可能性のあるエラーに備え、対処することです。ランタイムにおけるエラーのキャッチや報告、またそれらの条件を検証するため、4Dは包括的なサポートを提供しています。

エラー処理は次の2つの要望に応えます：

- 開発フェーズにおいて、問題となりうるコードのエラーやバグを発見して修正したい。
- 運用フェーズにおいて、予期しないエラーを検知して回復したい。とくに、システムエラーダイアログ（ディスクが一杯、ファイルがない、など）を独自のインターフェースに置換できます。

4D Server 上で実行されるコードのため、4D Server にはエラー処理メソッドを実装しておくことが強く推奨されます。このメソッドによって、サーバーマシンにおいて予期せぬダイアログが表示されることを防ぎ、エラーの調査に必要なログを専用ファイルにとることができます。

エラー/ステータス

`entity.save()` や `transporter.send()` など、おおくの 4D クラス関数は `status` オブジェクトを返します。ランタイムにおいて "想定される"、プログラムの実行を停止させないエラー（無効なパスワード、ロックされたエンティティなど）がこのオブジェクトに格納されます。これらのエラーへの対応は、通常のコードによっておこなうことができます。

ディスク書き込みエラーやネットワークの問題などのイレギュラーな中断は "想定されない" エラーです。これらのエラーは例外を発生させ、エラー処理メソッドを介して対応する必要があります。

エラー処理メソッドの実装

4Dにおいては、エラー専用のプロジェクトメソッドである エラー処理（または エラーキャッチ）メソッド内ですべてのエラーをキャッチし、処理することができます。

このプロジェクトメソッドはカレントプロセスに対して実装（インストール）され、インタープリターモードかコンパイルモードかにかかわらず、プロセス内で発生するすべてのエラーの際に自動で呼び出されます。このプロジェクトメソッドを 実装 するには、`ON ERR CALL` コマンドをコールし、コマンドに当該プロジェクトメソッド名を引数として渡します。たとえば：

```
ON ERR CALL("IO_ERRORS") // エラー処理メソッドを実装します
```

エラーの検知を中止するには、空の文字列を指定して再度 `ON ERR CALL` コマンドをコールします：

```
ON ERR CALL("") // エラーの検知を中止します
```

`Method called on error` コマンドは、`ON ERR CALL` によってカレントプロセスにインストールされているエラー処理メソッド名を返します。このコマンドは汎用的なコードでとくに有用です。エラー処理メソッドを一時的に変更し、後で復元することができます：

```
$methCurrent:=Method called on error
ON ERR CALL("NewMethod")
// ドキュメントを開くことができなければエラーが生成されます
$ref:=Open document("MyDocument")
// 前のエラー処理メソッドに戻します
ON ERR CALL($methCurrent)
```

スコープとコンポーネント

アプリケーションにおいて一つのエラーキャッチメソッドを使うやり方もあるれば、アプリケーションのモジュールごとに違うメソッドを定義する方法もあります。ただ

し、一つのプロセスにつき実装できるのは一つのメソッドのみです。

`ON ERR CALL` コマンドによって実装されたエラー処理メソッドは実行中のアプリケーションにしか適用されません。つまり、コンポーネント によってエラーが生成されても、ホストアプリケーションにおいて `ON ERR CALL` で実装されたエラー処理メソッドは反応しませんし、逆もまた然りです。

メソッド内でのエラー処理

独自に作成してエラー処理メソッド内では、エラーを調査するための情報がいくつか提供されています：

- 専用のシステム変数 (*):

- `Error` (倍長整数): エラーコード
- `Error method` (テキスト): エラーを生成したメソッドの名称
- `Error line` (倍長整数): エラーを生成したメソッドの行番号
- `Error formula` (テキスト): エラーの元となった 4D コードのフォーミュラ (テキスト)

(*) 4D は、いくつかの システム変数 と呼ばれる専用の変数を自動的に管理しています。 詳細については [4D ランゲージマニュアル](#) を参照ください。

- `GET LAST ERROR STACK` コマンドは、4D アプリケーションの現在のエラースタックに関する情報を返します。
- `Get call chain` コマンドは、カレントプロセス内における、メソッド呼び出しチェーンの各ステップを詳細に説明するオブジェクトのコレクションを返します。

例題

簡単なエラー処理システムの例です：

```
// エラー処理メソッドをインストールします
ON ERR CALL("errorMethod")
//... コードの実行
ON ERR CALL("") // エラーの検知を中止します
```

```
// errorMethod プロジェクトメソッド
If(Error#1006) // これはユーザーによる割り込みではありません
  ALERT("エラー "+String(Error)+" が発生しました。 問題となったコードはこちらです: \\"+Error formula+"\\")

End if
```

空のエラー処理メソッド

標準のエラーダイアログを表示させないようにするには、空のエラー処理メソッドを実装するだけで実現できます。`Error` システム変数はエラー処理メソッド以外のメソッドでも確認することができます：

```
ON ERR CALL("emptyMethod") // emptyMethod は空のエラー処理メソッドです
$doc:=Open document( "myFile.txt")
If (Error=-43)
  ALERT("ファイルが見つかりません。")
End if
ON ERR CALL("")
```

インタープリターモードとコンパイル済みモード

4D アプリケーションは インタープリター または コンパイル済み モードで実行することができます:

- インタープリターモードにおいて、コードは実行時に読み込まれてマシン語に翻訳されます。コードはいつでも追加・変更することができ、アプリケーションは自動的に更新されます。
- コンパイル済みモードにおいては、コンパイル時にすべてのコードが一括で読み込まれて翻訳されます。コンパイル後のアプリケーションにはアセンブリレベルの指示のみが残され、コード編集はできません。

コンパイルには以下のようなメリットがあります:

- 速度: アプリケーションの実行速度を3倍から1000倍速くします。
- コードチェック: アプリケーションコードの整合性をチェックし、論理的矛盾や構文的矛盾を検出します。
- 保護: アプリケーションをコンパイルすると、インターフリターコードを削除できます。コンパイルされたアプリケーションは、故意的にも不注意からもストラクチャーやメソッドの表示・修正ができないこと以外は、オリジナルのアプリケーションと同じに動作します。
- ダブルクリックで起動するアプリケーション: コンパイル後のアプリケーションは、独自のアイコンを持つスタンドアロンアプリケーション (.EXEファイル) に作り変えることもできます。
- プリエンプティブモードでの実行: プリエンプティブプロセスとして実行できるのは、コンパイルされたコードに限られます。

インターフリターコードとコンパイル済みコードの違い

アプリケーションの動作は同じであっても、インターフリターモードとコンパイル済みモードにはいくつかの相違点があり、コンパイルされるコードを書くにあたってはこれらを意識しておく必要があります。基本的に、4D のインターフリターはコンパイラーより柔軟です。

コンパイル済みコード	インターフリターコード
変数名とメソッド名が被ってはいけません。	エラーは生成されませんが、メソッドが優先されます。
コンパイラーアクション (例: C_LONGINT) によって、またはコンパイル時にコンパイラーアクションによって、すべての変数は型指定されなければなりません。	変数の型は実行中に決定していくことができます (推奨されません)
変数や配列のデータタイプは変更できません。	変数や配列のデータタイプは変更可能です (推奨されません)
1次元配列を2次元配列に、また2次元配列を1次元配列に変更することはできません。	可能です。
コンパイラーアクションにより変数のタイプ定義はおこなわれますが、フォーム上の変数のようにデータタイプが明確でない場合は、コンパイラーアクションを使用して変数のデータタイプを指定するべきです。	
Undefined 関数は、常に False を返します。変数は常に定義されています。	
メソッドの "プリエンプティブプロセスで実行可能" プロパティにチェックを入れていた場合、コードは他のスレッドアクションやメソッドを呼び出してはいけません。	プリエンプティブプロセスプロパティは無視されます。
特定のループの場合、割り込みを可能にするには IDLE コマンドが必要です。	いつでも割り込み可能です。

インターフリターでコンパイラーアクションを使用する場合

コンパイルしないアプリケーションには、コンパイラーアクションは必須ではありません。インターフリターは、各ステートメントにおける変数の使い方に準じて自動的に変数の型を設定し、アプリケーションプロジェクト内の変数の型を変えることができます。

インターフリターがこのように柔軟に対応するので、インターフリターモードとコンパイル済みモードでは、アプリケーションの動作が異なることがあります。

たとえば、あるところでは次のように記述して:

```
C_LONGINT(MyInt)
```

プロジェクトの他の場所では、下記のように記述した場合:

```
MyInt:=3.1416
```

この例では、コンパイラー指示子が代入ステートメントより 先に 解釈されれば、インタープリターでもコンパイル後でも `Myint` には同じ値 (3) が代入されます。

4D のインターパリターは、コンパイラー指示子を使用して変数の型を定義します。コード内でコンパイラー指示子が検出されると、インターパリターはそれに従って変数の型を定義します。それ以降のステートメントで間違った値を代入しようとすると (たとえば、数値変数に文字を割り当てるなど)、代入はおこなわれず、エラーが生成されます。

コンパイラーにとっては、この2つのステートメントのどちらが先に表示されても問題ではありません。はじめにプロジェクト全体を調べてコンパイラー指示子を探すからです。しかし、インターパリターは系統立てて処理するわけではなく、実行される順にステートメントを解釈します。ユーザーが何をおこなうかによって、この順序は当然異なります。したがって、常にコンパイラー指示子によって型定義してから変数を使用するようプロジェクトを計画することが肝心です。

ポインターの使用で型の矛盾を避ける

変数の型は変更することができません。しかし、ポインターを活用して異なるデータ型の変数を参照することはできます。たとえば、次のコードはインターパリターおよびコンパイル済みモードの両方で動作します：

```
C_POINTER($p)
C_TEXT($name)
C_LONGINT($age)

$name:="Smith"
$age:=50

$p:==>$name // テキストへのポインター
$p->:="Wesson" // テキストの代入

$p:==>$age
// 数値へのポインターに変更
$p->:=55 // 数値の代入
```

値の型に関わらず、その値の長さ (文字の数) を返す関数を考えてみましょう：

```
// Calc_Length 文字の数を返す関数
// $1 = 数値、テキスト、時間、ブール型の変数へのポインター

C_POINTER($1)
C_TEXT($result)
C_LONGINT($0)
$result:=String($1->)
$0:=Length($result)
```

この関数は次のように使えます：

```
$var1:="my text"
$var2:=5.3
$var3:=?10:02:24?
$var4:=True

$vLength:=Calc_Length(->$var1)+Calc_Length(->$var2)+Calc_Length (->$var3)+Calc_Length(->$var4)

ALERT("長さの合計: "+String($vLength))
```

コンポーネント

4D のコンポーネントとは、プロジェクトにインストール可能な、1つ以上の機能を持つ 4Dメソッドやフォームの一式です。たとえば、[4D SVGコンポーネント](#)は、SVGファイルの表示するための高度なコマンドと統合されたレンダリングエンジンを追加します。

コンポーネントの見つけ方

いくつかのコンポーネントは [4D開発環境にプリインストール](#)されていますが、4Dコミュニティによる多くの 4Dコンポーネントが [GitHub 上に公開](#)されています。また、[独自の 4Dコンポーネントを開発](#)することもできます。

コンポーネントのインストール

コンポーネントをインストールするには、[プロジェクトの Components フォルダー](#)にコンポーネントファイルをコピーします。エイリアスまたはショートカットも使用できます。

インタープリターモードで動作するホストプロジェクトは、インターパーまたはコンパイル済みどちらのコンポーネントも使用できます。コンパイルモードで実行されるホストデータベースでは、インターパーのコンポーネントを使用できません。この場合、コンパイル済みコンポーネントのみが利用可能です。

コンポーネントの使い方

コンポーネントメソッドやフォームは、4D開発において標準の要素として使用できます。

インストールされたコンポーネントにメソッドが含まれている場合、それらはエクスプローラーのメソッドページの [コンポーネントメソッド](#) テーマに表示されます。

コンポーネントメソッドを選択し、エクスプローラーのドキュメント ボタンをクリックすると、そのメソッドに関する情報が得られます ([あれば](#))。

The screenshot shows the 4D Explorer window titled "myProject - Explorateur". The left sidebar lists categories: Démarrage, Tables, Formulaires, Méthodes (selected), Commandes, Constantes, Plug-ins, and Corbeille. The main pane displays the "Méthodes" section under "Méthodes composant". A tree view shows various components like 4D Labels, 4D Mobile App, and 4D Mobile App Server, with "PushNotification" selected under "4D Mobile App Server". To the right, a detailed view for "PushNotification" is shown with a yellow bell icon. The title is "PushNotification". Below it is a description: "Utility class to send a push notification to one or multiple recipients." A section titled "Usage" provides instructions: "In order to use the component to send push notification, it is required to have an authentication key file AuthKey_XXXXXX.p8 from Apple." It also suggests placing the file in the application sessions folder ("MobileApps/TEAM123456.com.sample.myappname"). Below this is a code snippet:

```
$pushNotification:=MobileAppServer .PushNotification.new()  
  
$notification:=New object  
$notification.title:="This is title"
```

At the bottom of the window are buttons for "+", "-", settings, preview, and documentation.

プラグイン

4D アプリケーションの開発を進めていくと、最初は気付かなかった数多くの機能を発見することでしょう。それだけでなく、プラグインを4D開発環境に追加することで、標準の4Dの機能を高めることもできます。

プラグインとは何か

プラグインとは、C や C++ などの言語で書かれた、4D 起動時にロードされるコードのことです。プラグインは、4D に機能を追加します。プラグインは、4D に機能を追加します。プラグインには通常複数のルーチンが含まれています。プラグインは外部エリアを操作でき、外部プロセスを実行できます。

プラグインの見つけ方

4D ユニティによって多数のプラグインが作成されています。公開されたプラグインは [GitHub](#) で確認できます。また、[独自のプラグインを開発](#) することもできます。

プラグインのインストール

4D 環境にプラグインをインストールするには、プラグインファイルを [Project フォルダーと同じ階層](#) の Plugins フォルダーにコピーします。

プラグインは 4D 起動時にロードされるので、これらをインストールする際には 4D アプリケーションを終了する必要があります。プラグインの利用に特別なライセンスが必要な場合、プラグインはロードされますが、ライセンスをインストールするまで使用することはできません。

プラグインの使い方

プラグインコマンドは、4D 開発において通常の4Dコマンドと同様に使用できます。プラグインコマンドは、エクスプローラーの プラグイン ページに表示されます。

識別子の命名規則

この章では、4D ランゲージにおけるさまざまな要素（変数、オブジェクトプロパティ、テーブル、フォームなど）の命名規則について説明します。

非ローマ文字（日本語など）が識別子に使用された場合、その最大長は短くなることがあります。

配列

変数と同じルールが適用されます。

クラス

クラス名は31文字以内で指定します。

クラス名は、ドット記法のための標準的な [プロパティ名の命名規則](#) に準拠している必要があります。

競合防止のため、[データベーステーブル](#) 同じ名前のクラスを作成するのは推奨されないこと

関数

関数名は、ドット記法のための標準的な [プロパティ名の命名規則](#) に準拠している必要があります。

Tip: アンダースコア ("_") 文字で関数名を開始すると、その関数は 4D コードエディターの自動補完機能から除外されます。

オブジェクトプロパティ

プロパティ名（オブジェクト 属性 とも呼びます）は255文字以内の文字列で指定します。

オブジェクトプロパティは、スカラー値・ORDA要素・クラス関数・他のオブジェクト等を参照できます。参照先に関わらず、[ドット記法](#) を使用するにはオブジェクトプロパティ名は次の命名規則に従う必要があります：

- 1文字目は、文字、アンダースコア(_)、あるいはドル記号 (\$) でなければなりません。
- その後の文字には、文字・数字・アンダースコア(_)・ドル記号 (\$) が使用できます。
- 大文字・小文字は区別されます。

例:

```
myObject.myAttribute:="10"  
$value:=$clientObj.data.address.city
```

大カッコ [] に文字列を含める記法を利用すれば、プロパティ名にはあらゆる文字を使用することができます（例： myObject["1. First property"] ）。

詳細は [ECMA Script standard](#) を参照ください。

引数

引数名は必ず \$ 文字で始まります。また、変数名と同じルールが適用されます。

例:

```
Function getArea($width : Integer; $height : Integer) -> $area : Integer
```

```
#DECLARE ($i : Integer ; $param : Date) -> $myResult : Object
```

プロジェクトメソッド

プロジェクトメソッド名は 31文字以内で指定します。

- 1文字目は、文字、数字、あるいはアンダースコア(_) でなければなりません。
- その後の文字には、文字・数字・アンダースコア(_)・スペースが使用できます。
- 予約語を使用しないでください。予約語にはコマンド名 (Date , Time 等)、キーワード (If , For 等)、そして定数 (Euro , Black , Friday 等) が含まれます。
- 大文字・小文字は区別されます。

例:

```
If(New client)
DELETE DUPLICATED VALUES
APPLY TO SELECTION([Employees];INCREASE SALARIES)
```

Tip: 4Dのビルトインコマンドと同じ命名規約を利用することは良いプログラミングテクニックです。メソッド名には大文字を使用しますが、メソッドが値を返す場合には最初の文字だけを大文字にします。このように命名することにより、数ヶ月後に保守のためプロジェクトを再度開いたときに、エクスプローラーウィンドウでその名前を見ただけで、メソッドが結果を返すかどうかがわかります。

メソッドを呼び出すには、メソッド名を書きます。しかし ON EVENT CALL など一部の 4Dのビルトインコマンドやプラグインコマンドに、引数としてメソッド名を渡す場合には、文字列 (ダブルクオートで括る) として渡します。

例:

```
// このコマンドはメソッド（関数）またはフォーミュラを受け取ります
QUERY BY FORMULA([aTable];Special query)
// このコマンドはメソッド（プロシージャ）またはステートメントを受け取ります
APPLY TO SELECTION([Employees];INCREASE SALARIES)
// このコマンドはメソッド名を文字列で受け取ります
ON EVENT CALL("HANDLE EVENTS")
```

テーブルとフィールド

大カッコ内 ([...]) に名前を入れることで、テーブルを表します。フィールドを表すには、そのフィールドが属するテーブルをまず指定し、フィールド名を続けて書きます。

テーブル名およびフィールド名は、31文字以内で指定します。

- 1文字目は、文字、アンダースコア(_)、あるいはドル記号 (\$) でなければなりません。
- その後の文字には、半角アルファベット文字・数字・スペース・アンダースコアを使用ができます。
- 予約語を使用しないでください。予約語にはコマンド名 (Date , Time 等)、キーワード (If , For 等)、そして定数 (Euro , Black , Friday 等) が含まれます。
- SQLで処理する場合には追加のルールがあります: 文字 _0123456789abcdefghijklmnopqrstuvwxyz のみを使用できます。また、名前に SQLキーワード (コマンド、属性 等) が含まれていてはなりません。

例:

```
FORM SET INPUT([Clients];"Entry")
ADD RECORD([Letters])
[Orders]Total:=Sum([Line]Amount)
QUERY([Clients];[Clients]Name="Smith")
[Letters]Text:=Capitalize text([Letters]Text)
```

競合防止のため、[クラス](#)と同じ名前のテーブルを作成するのは推奨されません。

変数

変数名は、スコープ記号 (\$ および <>) を除いて最大31文字以内で指定することができます。

- 変数名の1文字目は、文字あるいはアンダースコア(_) でなければなりません。また、[引数](#) や [ローカル変数](#) の場合はドル記号 (\$)、[インターフォラム変数](#) の場合はインタープロセス記号 (<>) を1文字目に使用します。
- 変数の1文字目に数字を使うことは許可されていますが、推奨されません。また、[var 宣言のシンタックス](#) ではサポートされていません。
- その後の文字には、文字・数字・アンダースコア(_) が使用できます。
- 変数名にスペースを使うことは許可されていますが、推奨されません。また、[var 宣言のシンタックス](#) ではサポートされていません。
- 予約語を使用しないでください。予約語にはコマンド名 ([Date](#), [Time](#) 等)、キーワード ([If](#), [For](#) 等)、そして定数 ([Euro](#), [Black](#), [Friday](#) 等) が含まれます。
- 変数名においては、大文字・小文字は区別されません。

例:

```
For($vlRecord;1;100) // ローカル変数
$vsMyString:="Hello there" // ローカル変数
var $vName; $vJob : Text // ローカル変数
If(bValidate=1) // プロセス変数
<>vlProcessID:=Current process() // インターフォラム変数
```

その他の識別子

4Dランゲージにおいては、識別子が文字列として扱われる要素も多数存在します：フォーム、フォームオブジェクト、命名セレクション、プロセス、セット、メニューバー、等。

このような文字列名は、(該当する場合はスコープ記号を除いて) 255文字以内で指定します。

- 文字列名にはあらゆる文字を使用できます。
- 文字列名においては、大文字・小文字は区別されません。

例:

```
DIALOG([Storage];"Note box"+String($vlStage))
OBJECT SET FONT(*;"Binfo";"Times")
USE NAMED SELECTION([Customers];"Closed")// プロセス命名セレクション
USE NAMED SELECTION([Customers];"<>ByZipcode") // インターフォラム変数命名セレクション
    // グローバルプロセス "Add Customers" の開始:
$vlProcessID:=New process("P_ADD_CUSTOMERS";48*1024;"Add Customers")
    // ローカルプロセス "$Follow Mouse Moves" の開始:
$vlProcessID:=New process("P_MOUSE_SNIFFER";16*1024;"$Follow Mouse Moves")
CREATE SET([Customers];"Customer Orders")// プロセスセット
USE SET("<>Deleted Records") // インターフォラム変数セット
If(Records in set("$Selection"+String($i))>0) // クライアントセット
```


データモデルオブジェクト

ORDA は、下地であるデータベースストラクチャへの自動マッピングに基づいた技術です。ORDA は、エンティティやエンティティセレクションオブジェクトを介してデータへのアクセスも提供します。結果的に ORDA は、データモデルオブジェクト一式の形でデータベース全体を公開します。

ストラクチャーマッピング

`ds` および `Open datastore` コマンドを使ってデータストアを呼び出すと、戻り値の **データストア オブジェクト**には、対応する 4D ストラクチャのテーブルとフィールドへの参照が属性として格納されています:

- テーブルはデータクラスへとマップされます。
- フィールドはストレージ属性へとマップされます。
- リレーションはリレーション属性へとマップされます。ストラクチャーエディター内で定義されたリレーション名はリレーション属性名として使用されます。

The screenshot shows the 4D Inspector window with the following details:

- Definition:**
 - From: [Project]companyID
 - To: [Company]ID
 - Color: Automatic
- Many to One Options:**
 - Name: theClient (highlighted with a red circle)
 - Manual
 - Auto Wildcard support
 - Wildcard Choice:
 - ID
 - name
 - discount
 - Prompt if related one does not exist
- One to Many Options:**
 - Name: companyProjects (highlighted with a red circle)
 - Manual
 - Auto assign related value in subform
- Deletion Control:**
 - SQL

Expression and **Value** sections show the mapped fields:

Expression	Value
ds.Company	DataClass: Company ("name": "companyProjects", "kind": "relatedEntities", "relatedDataClass": "Project", "type": "ProjectSelection") ("name": "discount", "kind": "storage", "type": "number", "fieldNumber": 3) ("name": "ID", "kind": "storage", "type": "long", "fieldNumber": 1) ("name": "name", "kind": "storage", "type": "string", "fieldNumber": 2)
ds.Project	DataClass: Project ("name": "companyId", "kind": "storage", "type": "long", "fieldNumber": 3) ("name": "ID", "kind": "storage", "type": "long", "fieldNumber": 1) ("name": "name", "kind": "storage", "type": "string", "fieldNumber": 2)
ds.Project.theClient	("name": "theClient", "kind": "relatedEntity", "relatedDataClass": "Company", "type": "Company")

変換のルール

変換の際には以下のルールが適用されます:

- テーブル、フィールド、そしてリレーション名はオブジェクトプロパティ名へとマップされます。それらの名前が標準のオブジェクト命名規則に則っているようにしてください ([識別子の命名規則](#) 参照)。
- データストアは単一のプライマリーキーを持つテーブルのみを参照します。以下のテーブルは参照されません:
 - プライマリーキーがないテーブル
 - 複合プライマリーキーを持つテーブル
- BLOBフィールドは、[BLOB オブジェクト](#) 型の属性として利用可能です。

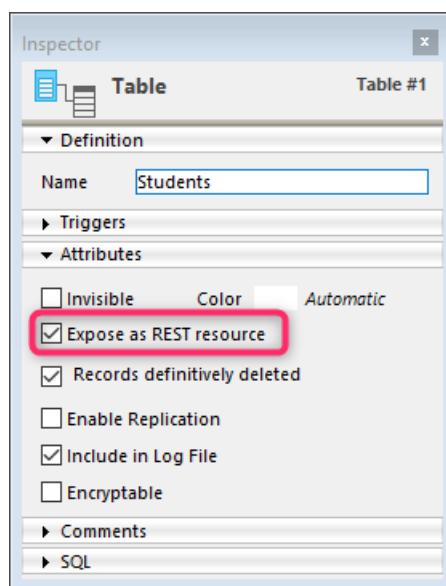
ORDA のデータストアマッピングでは、次のものは考慮されません:

- テーブルあるいはフィールドの"非表示"オプション
 - `SET TABLE TITLES` あるいは `SET FIELD TITLES` を通して定義されたバーチャルストラクチャー
 - リレーションの"手動"あるいは"自動"プロパティ

リモートデータストアの利用

`Open datastore` コマンドまたは [REST リクエスト](#) によってリモートデータストアにアクセスする場合、RESTリソースとして公開 プロパティが設定されているテーブルとフィールドのみ利用可能です。

このプロパティは、データストアにデータクラスおよび属性として公開したい各テーブルおよびフィールドについて 4D ストラクチャーのレベルで設定する必要があります:



データモデルのアップデート

データベースストラクチャーレベルで変更がおこなわれると、カレントの ORDA モデルレイヤーは無効化されます。これらの変更には、以下のものが含まれます:

- テーブル、フィールド、リレーションの追加または削除
- テーブル、フィールド、リレーションの名称変更
- フィールドの核となるプロパティ (型、重複不可、インデックス、自動インクリメント、null値サポートなど) の変更

カレントの ORDA モデルレイヤーが無効化されると、その後 4D または 4D Server のローカルの `ds` データストアを呼び出した時にモデルレイヤーが自動的に再読み込みされ、更新されます。ただし、エンティティやエンティティセレクションなど、ORDA オブジェクトへの既存の参照は、再生成されるまではそれらが作成されたときのモデルを使用し続けるという点に注意してください。

また、アップデートされた ORDA モデルレイヤーは、以下のコンテキストにおいては自動的には利用可能にはなりません:

- 4D Server に接続したリモートの 4D -- リモートのアプリケーションはサーバーに再接続する必要があります
- `Open datastore` または `REST 呼び出し` を使用して開かれたリモートデータストア -- 新しいセッションを開く必要があります

オブジェクトの定義

データストア

データストアは、データベースへのインターフェースオブジェクトです。データベース全体を反映したものをオブジェクトとしてビルドします。データストアは モデルと データ から構成されています:

- モデルにはデータストアを構成するすべてのデータクラスが格納され、その詳細な情報も含まれます。これはその下地にあるデータベース自体からは独立した存在です。
- データとは、そのモデル内で使用・保存される情報を指します。たとえば、従業員の名前、住所、生年月日などはデータストア内で扱うことができるデータに含まれます。

コード内で扱うにあたっては、データストアはオブジェクトであり、公開されているすべての `データクラス` をプロパティとして持ります。

4D では次のデータストアを扱うことができます:

- カレント 4D データベースに基づいた、ローカルデータストア。これは、`ds` コマンドで返されるメインデータストアです。
- リモートデータベースによって REST リソースとして公開された、一つ以上のリモートデータストア。これらは、`Open datastore` コマンドで返されます。

データストアは単一の、ローカルあるいはリモートのデータベースを参照します。

データストアオブジェクト自身は、オブジェクトとしてコピーすることはできません:

```
$mydatastore:=OB Copy(ds) // null を返します
```

しかしながらデータストアプロパティは取得可能です:

```
ARRAY TEXT($prop;0)
OB GET PROPERTY NAMES(ds;$prop)
// $prop にはすべてのデータクラスの名前が格納されます
```

メイン（デフォルト）のデータストアは `ds` コマンドを通して常に利用可能です。 `Open datastore` コマンドを使えば、あらゆるリモートデータストアを参照することができます。

データクラス

データクラスとは、テーブルに相当するものです。オブジェクトモデルとして使用され、リレーション属性（データクラス間のリレーションに基づいてビルトされた属性）を含めてすべてのフィールドを属性として参照します。リレーション属性はクエリにおいて通常の属性のように使用することができます。

4D プロジェクト内のすべてのデータクラスは、`ds` データストアのプロパティとして利用可能です。 `Open datastore` コマンドまたは [REST リクエスト](#) によってアクセスするリモートデータストアの場合、データストアのデータクラスとして公開したい各テーブルについて 4D ストラクチャーのレベルで REST リソースとして公開 プロパティを設定する必要があります。

たとえば、4D ストラクチャー内の以下のテーブルについて考えます。

Company	
ID	2 ³²
name	A
creationDate	17
revenues	0.5
extra	{ }

`Company` テーブルは `ds` データストア内のデータクラスとして自動的に利用可能です。以下のように書くことができます:

```
var $compClass : cs.Company // Company クラスのオブジェクト変数として $compClass を宣言します
$compClass:=ds.Company // Company データクラスへの参照を $compClass に代入します
```

データクラスオブジェクトは以下のものを格納することができます:

- attributes
- リレーション属性

データクラスは実際のデータベースの概略を提供し、概念的なデータモデルの管理を可能にします。データクラスはデータストアをクエリする唯一の方法です。クエリは単一のデータクラスを通して実行されます。クエリはデータクラスの属性およびリレーション属性名に基づいてビルトされます。リレーション属性は、一つのクエリ内で複数のリンクされたテーブルを用いる手段です。

データクラスオブジェクト自身は、オブジェクトとしてコピーすることはできません:

```
$mydataclass:=OB Copy(ds.Employee) // null を返します
```

しかしながらデータクラスプロパティは取得可能です:

```
ARRAY TEXT($prop;0)
OB GET PROPERTY NAMES(ds.Employee;$prop)
// $prop にはすべてのデータクラス属性の名前が格納されます
```

属性

データクラスプロパティは、下地にあるフィールドやリレーションを説明する属性オブジェクトです。たとえば:

```
$nameAttribute:=ds.Company.name // クラス属性への参照  
$revenuesAttribute:=ds.Company["revenues"] // 別の書き方
```

このコードは、`$nameAttribute` および `$revenuesAttribute` に、`Company` クラスの `name` および `revenues` 属性の参照をそれぞれ代入します。このシンタクスは属性内に保管されている値を返すではありません。その代わりに、属性自身への参照を返します。値を管理するために [エンティティ](#) を使用する必要があります。

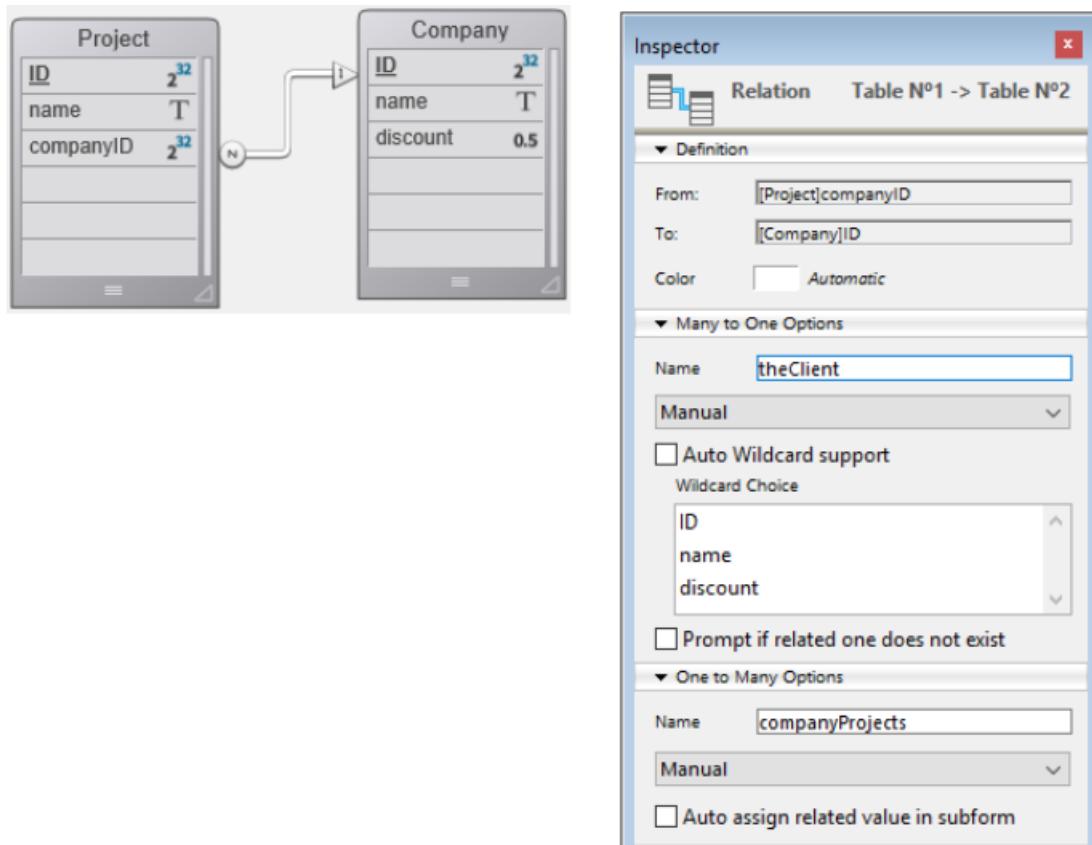
テーブル内の適格なフィールドはすべて、親 [データクラス](#) の属性として利用可能です。`Open datastore` コマンドまたは [REST リクエスト](#) によってアクセスするリモートデータストアの場合、データクラスの属性として公開したい各フィールドについて 4D ストラクチャーのレベルで RESTリソースとして公開プロパティを設定する必要があります。

ストレージ属性とリレーション属性

データクラス属性にはいくつかの種類があります。ストレージ、リレートエンティティ、リレートエンティティズです。スカラーである属性（単一の値のみを提供するもの）は標準の 4D データ型（整数、テキスト、オブジェクトなど）をサポートします。

- **ストレージ属性** は4D データベース内のフィールドに相当するもので、インデックスをつけることができます。ストレージ属性に割り当てられた値は、保存時にエンティティの一部として保存されます。ストレージ属性にアクセスしたとき、その値はデータストアから直接取り出されます。ストレージ属性はエンティティを構成するもっとも基礎的な要素であり、名前とデータ型により定義されます。
- **リレーション属性** は他のエンティティへのアクセスを提供します。リレーション属性は単一のエンティティ（あるいはエンティティなし）あるいはエンティティセレクション（0からNまでのエンティティ）のどちらかになります。リレーション属性はリレーションナルストラクチャーの "クラシックな" リレーションに基づいており、リレートエンティティあるいはリレートエンティティズへの直接的なアクセスを提供します。リレーション属性は、ORDAにおいては名前を使用することで直接的に利用可能です。

たとえば、以下の部分的なデータベースストラクチャーと、そのリレーションプロパティについて考えます:



すべてのストレージ属性は自動的に利用可能です:

- `Project` データクラス内: "ID", "name", および "companyID"
- `Company` データクラス内: "ID", "name", および "discount"

これに加えて、以下のリレーション属性もまた自動的に利用可能になります:

- Project データクラス内: "リレートエンティティ" 型の theClient 属性。各 Project (クライアント) に対して最大 1つの Companyがあります。
- Company データクラス内: "リレートエンティティズ" 型の companyProjects 属性。各 Company に対して不定数の Project があります。

データベースリレーションの手動あるいは自動プロパティは、ORDAにおいては何の効力も持ちません。

すべてのデータクラス属性はデータクラスのプロパティとして公開されています:

Expression	Value
ds.Company	DataClass: Company
ds.Company.companyProjects	{"name":"companyProjects","kind":"relatedEntities","relatedDataClass":"Project","type":"ProjectSelection"} {"name":"discount","kind":"storage","type":"number","fieldNumber":3} {"name":"ID","kind":"storage","type":"long","fieldNumber":1} {"name":"name","kind":"storage","type":"string","fieldNumber":2}
ds.Company.discount	
ds.Company.ID	
ds.Company.name	
ds.Project	DataClass: Project
ds.Project.companyID	{"name":"companyID","kind":"storage","type":"long","fieldNumber":3} {"name":"ID","kind":"storage","type":"long","fieldNumber":1} {"name":"name","kind":"storage","type":"string","fieldNumber":2}
ds.Project.ID	
ds.Project.name	
ds.Project.theClient	{"name":"theClient","kind":"relatedEntity","relatedDataClass":"Company","type":"Company"}

これらのオブジェクトは属性を表しますが、データへのアクセスは与えないという点に注意してください。データの読み書きは [エンティティオブジェクト](#) を通じておこなわれます。

計算属性

計算属性は、[Entity クラスの定義](#) で `get <attributeName>` 関数を使用して宣言されます。この属性値はアクセスされるたびに評価され、保存されません。計算属性は、基礎となるデータベースストラクチャーには属しませんが、その上に構築され、データモデルの属性と同様に使用することができます。

エンティティ

エンティティとは、レコードに相当するものです。実際にはデータベース内のレコードを参照するオブジェクトです。エンティティは、[データクラス](#) のインスタンスとも解釈可能なオブジェクトです。同時にエンティティは、データストアがもとにしているデータベースに相關するデータも格納しています。

エンティティの目的はデータの管理 (作成、更新、削除) です。エンティティセレクションを用いてエンティティ参照を取得した場合、その参照にはエンティティセレクションについての情報も保持されるため、セレクションを走査することができます。

エンティティオブジェクト自身は、オブジェクトとしてコピーすることはできません:

```
$myentity:=OB Copy(ds.Employee.get(1)) // null を返します
```

しかしながらエンティティプロパティは取得可能です:

```
ARRAY TEXT($prop;0)
OB GET PROPERTY NAMES(ds.Employee.get(1);$prop)
// $prop にはすべてのエンティティ属性の名前が格納されます
```

エンティティセレクション

エンティティセレクションとは、同じデータクラスに所属する一つ以上のエンティティへの参照を格納しているオブジェクトのことです。通常、クエリの結果として、あるいはリレーション属性の戻り値として作成されます。エンティティセレクションは、データクラスから 0個、1個、あるいは X個のエンティティを格納することができます (X はデータクラスに格納されているエンティティの総数です)。

例:

```
var $e : cs.EmployeeSelection // EmployeeSelection クラスのオブジェクト変数として $e を宣言します
$e:=ds.Employee.all() // 結果のエンティティセレクションへの参照を $e に代入します
```

エンティティセレクションは "順列あり" あるいは "順列なし" 状態のどちらかです ([後述参照](#))。

また、[どのように作成されたか](#) に応じて、エンティティセレクションは "共有可能" あるいは "追加可能" 状態のどちらかです。

エンティティセレクションオブジェクト自身は、オブジェクトとしてコピーすることはできません：

```
$myentitysel:=OB Copy(ds.Employee.all()) // null を返します
```

しかしながらエンティティセレクションプロパティは取得可能です：

```
ARRAY TEXT($prop;0)
OB GET PROPERTY NAMES(ds.Employee.all();$prop)
// $prop にはエンティティセレクションのプロパティ名が格納されます
// ("length", "00", "01"...)
```

エンティティセレクションの順列あり/順列なし

`orderBy()` メソッドを使用した場合、あるいは特定のオプションを使用した場合は除き、4D ORDA は最適化の観点からデフォルトで順列なしのエンティティセレクションを作成します。このドキュメントでは、指定されている場合を除き、"エンティティセレクション" は "順列なしのエンティティセレクション" を指すこととします。

順列ありのエンティティセレクションは、必要な場合において、あるいはオプションを使用して特別に要求した場合に限り作成されます。たとえば、以下のような場合です：

- セレクション (タイプを問わず) に対して、あるいはデータクラスに対して `orderBy()` を使った場合の戻り値
- `newSelection()` メソッドに `dk keep ordered` オプションを渡した場合の戻り値

順列なしのエンティティセレクションは以下のような場合に作成されます：

- セレクション (タイプを問わず) に対して、あるいはデータクラスに対して標準の `query()` を使った場合の戻り値
- オプションなしで `newSelection()` メソッドを使用した場合の戻り値
- 任意の演算メソッド (`or()`, `and()`, `minus()`) を使った場合の戻り値 (入力セレクションタイプは問いません)。

次のエンティティセレクションは常に 順列あり となります。

- 4D Server からリモートクライアントに返されるエンティティセレクション
- リモートデータストアにおいて作成されるエンティティセレクション

順列ありのエンティティセレクションが順列なしのエンティティセレクションになった場合、重複したエンティティ参照はすべて削除されます。

データモデルクラス

ORDA を使用して、データモデル上に高レベルクラス関数を作成することができます。これによってビジネス指向のコードを書き、APIのように "公開" することができます。データストア、データクラス、エンティティ、およびエンティティセレクションはそれぞれ、関数を持つことのできるクラスオブジェクトとして提供されています。

たとえば、選択中の社員より給与の高い社員一覧を返す `getNextWithHigherSalary()` 関数を `EmployeeEntity` クラスに作成したとします。この関数は簡単に呼び出すことができます：

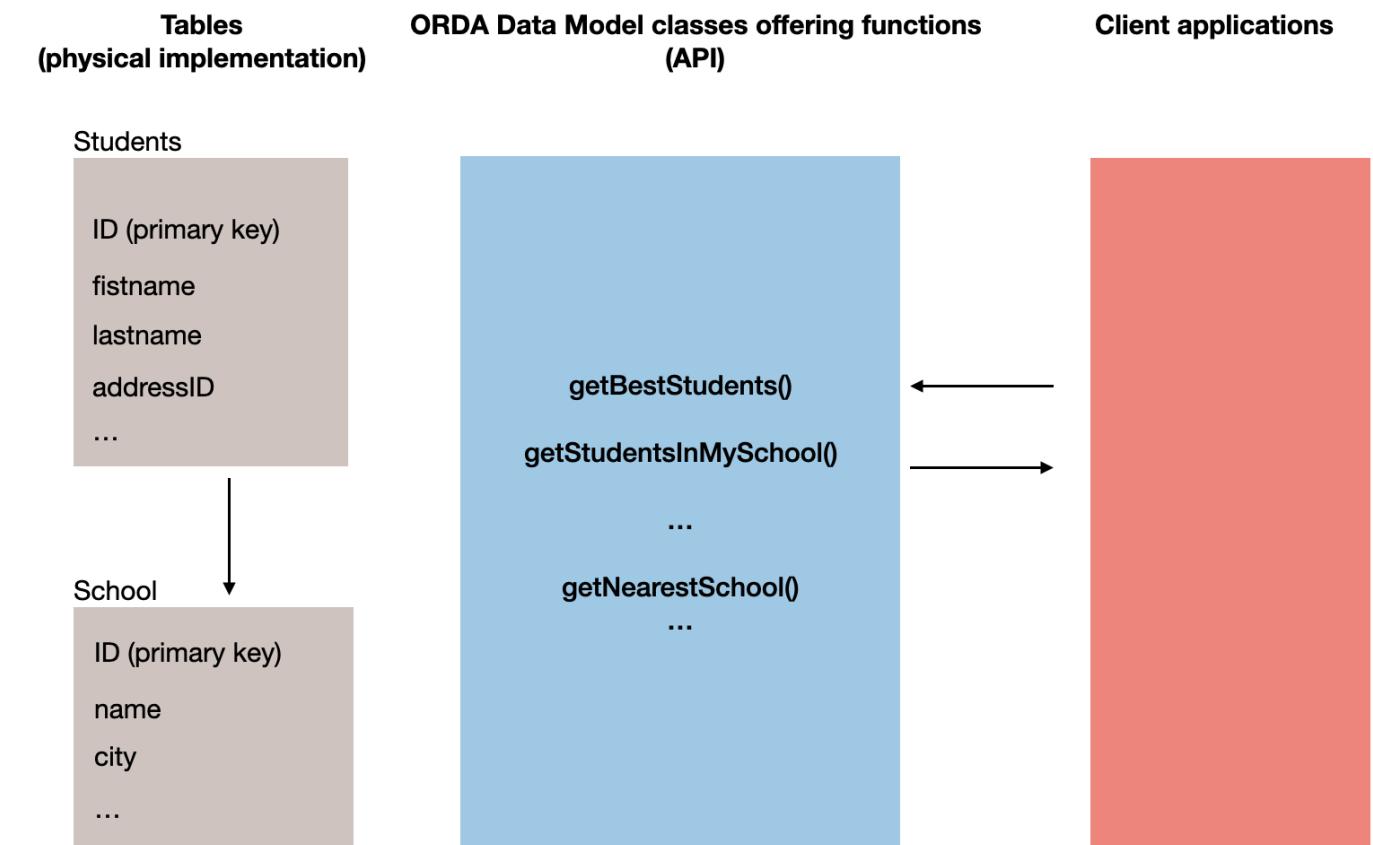
```
$nextHigh:=ds.Employee(1).getNextWithHigherSalary()
```

これらの関数はローカルデータストアだけでなく、クライアント/サーバーやリモートアーキテクチャーでも使用することができます：

```
//$cityManager はリモートデータストアへの参照です  
Form.comp.city:=$cityManager.City.getCityName(Form.comp.zipcode)
```

この機能により、4D アプリケーションのビジネスロジックをまるごと独立したレイヤーに保存し、高レベルのセキュリティで簡単に管理・利用することができます：

- わかりやすく使いやすい関数のみを公開し、その裏にある構造の複雑性を "隠す" ことができます。
- 構造が発展した場合には影響を受ける関数を適応させるだけで、クライアントアプリケーションは引き続き透過的にそれらを呼び出すことができます。
- デフォルトでは、データモデルクラス関数（計算属性関数 含む）および エイリアス属性 はすべて、リモートアプリケーションに対して 非公開 に設定されており、RESTリクエストで呼び出すことはできません。公開する関数やエイリアスは `exposed` キーワードによって明示的に宣言する必要があります。

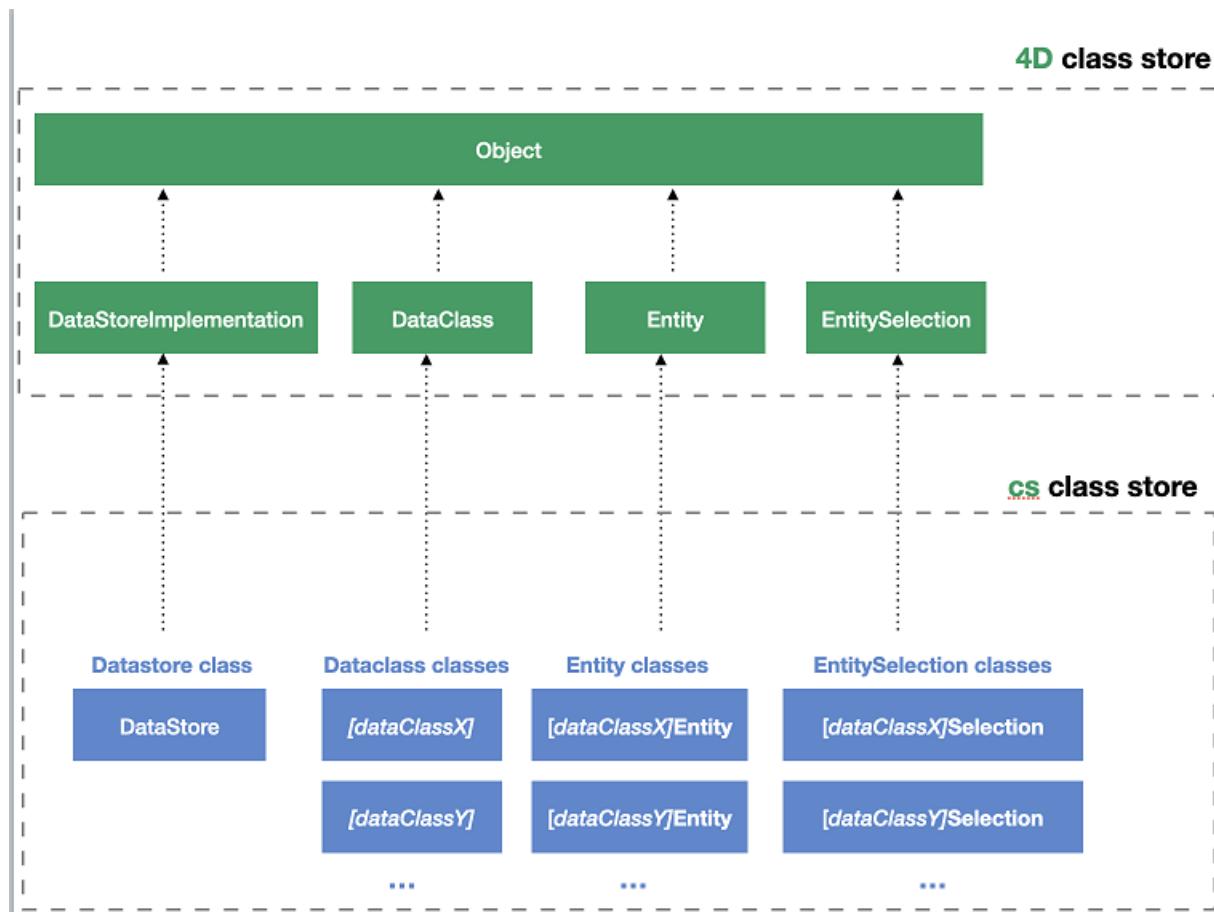


4D database (eventually exposed as a REST server)

各データモデルオブジェクトに関わるクラスは、4D によって [あらかじめ自動的に作成](#) されます。

アーキテクチャー

ORDA では、4D クラスストアを介して公開される 汎用クラス と、cs クラスストアで公開される ユーザークラス が提供されています：



ORDA データモデルクラスはすべて cs クラスストアのプロパティとして公開されます。次の ORDA クラスが提供されています：

クラス	例	次によってインスタンス化されます
cs.DataStore	cs.DataStore	<code>ds</code> コマンド
cs.DataClassName	cs.Employee	<code>dataStore.DataClassName</code> , <code>dataStore["DataClassName"]</code>
cs.DataClassNameEntity	cs.EmployeeEntity	<code>dataClass.get()</code> , <code>dataClass.new()</code> , <code>entitySelection.first()</code> , <code>entitySelection.last()</code> , <code>entity.previous()</code> , <code>entity.next()</code> , <code>entity.first()</code> , <code>entity.last()</code> , <code>entity.clone()</code>
cs.DataClassNameSelection	cs.EmployeeSelection	<code>dataClass.query()</code> , <code>entitySelection.query()</code> , <code>dataClass.all()</code> , <code>dataClass.fromCollection()</code> , <code>dataClass.newSelection()</code> , <code>entitySelection.drop()</code> , <code>entity.getSelection()</code> , <code>entitySelection.and()</code> , <code>entitySelection.minus()</code> , <code>entitySelection.or()</code> , <code>entitySelection.orderBy()</code> , <code>entitySelection.orderByFormula()</code> , <code>entitySelection.slice()</code> , Create entity selection

ORDA ユーザークラスは通常のクラスファイル (.4dm) としてプロジェクトの Classes サブフォルダーに保存されます ([後述参照](#))。

ORDA データモデルユーザークラスのオブジェクトインスタンスは、それらの親クラスのプロパティや関数を使うことができます：

- Datastore クラスオブジェクトは、[ORDA Datastore 汎用クラス](#) の関数を呼び出すことができます。

- DataClass クラスオブジェクトは、[ORDA DataClass 汎用クラス](#) の関数を呼び出すことができます。
- EntitySelection クラスオブジェクトは [ORDA EntitySelection 汎用クラス](#) の関数を呼び出すことができます。
- Entity クラスオブジェクトは [ORDA Entity 汎用クラス](#) の関数を呼び出すことができます。

クラスの説明

▶ [履歴](#)

DataStore クラス

4D のデータベースは、自身の DataStore クラスを `cs` クラスストアに公開します。

- 親クラス: `4D.DataStoreImplementation`
- クラス名: `cs.DataStore`

DataStore クラス内には、`ds` オブジェクトを介して使用する関数を作成することができます。

例題

```
// cs.DataStore class

Class extends DataStoreImplementation

Function getDesc
$0:="社員と会社を公開するデータベース"
```

この関数は次のように使えます:

```
$desc:=ds.getDesc() // "社員と会社を..."
```

DataClass クラス

ORDA で公開されるテーブル毎に、DataClass クラスが `cs` クラスストアに公開されます。

- 親クラス: `4D.DataClass`
- クラス名: `cs.DataClassName` (`DataClassName` はテーブル名です)
- 例: `cs.Employee`

例題

```
// cs.Company クラス

Class extends DataClass

// 収益が平均以上の会社を返します
// Company DataClass にリレートしているエンティティセレクションを返します

Function GetBestOnes()
$sel:=This.query("revenues >= :1";This.all().average("revenues"));
$0:=$sel
```

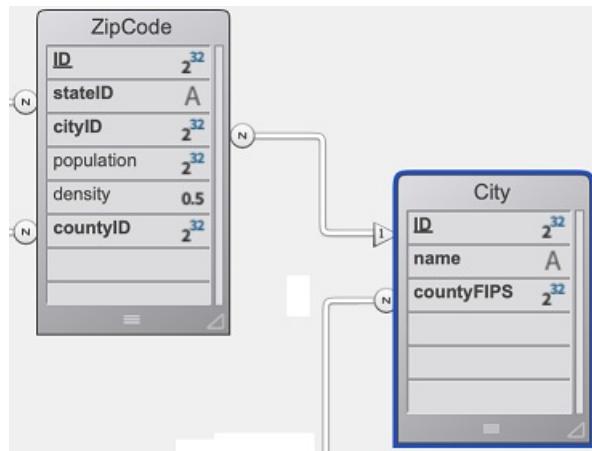
全会社データから平均以上の会社データをエンティティセレクションに抽出するには次を実行します:

```
var $best : cs.CompanySelection
$best:=ds.Company.GetBestOnes()
```

計算属性は Entity クラスにおいて定義されます。

リモートデータストアの例

次の City カタログをリモートデータストアとして公開しています:



City クラス は API を提供しています:

```
// cs.City クラス  
  
Class extends DataClass  
  
Function getCityName()  
    var $1; $zipcode : Integer  
    var $zip : 4D.Entity  
    var $0 : Text  
  
    $zipcode:=$1  
    $zip:=ds.ZipCode.get($zipcode)  
    $0:=""  
  
    If ($zip#Null)  
        $0:=$zip.city.name  
    End if
```

クライアントはまず、リモートデータストアのセッションを開始します:

```
$cityManager:=Open datastore(New object("hostname";"127.0.0.1:8111");"CityManager")
```

クライアントアプリケーションは API を使い、たとえばフォームに入力された郵便番号 (zipcode) に合致する都市を取得することができます:

```
Form.comp.city:=$cityManager.City.getCityName(Form.comp.zipcode)
```

EntitySelection クラス

ORDA で公開されるテーブル毎に、EntitySelection クラスが cs クラスストアに公開されます。

- 親クラス: 4D.EntitySelection
- クラス名: *DataClassNameSelection* (*DataClassName* はテーブル名です)
- 例: cs.EmployeeSelection

例題

```
// cs.EmployeeSelection クラス

Class extends EntitySelection

// 給与が平均超えの社員を当該エンティティセレクションから抽出します

Function withSalaryGreaterThanAverage
    C_OBJECT($0)
    $0:=This.query("salary > :1"; This.average("salary")).orderBy("salary")
```

任意の社員エンティティセレクションより、給与が平均以上の社員を取得するには:

```
$moreThanAvg:=ds.Company.all().employees.withSalaryGreaterThanAverage()
```

Entity クラス

ORDA で公開されるテーブル毎に、Entity クラスが `cs` クラストアに公開されます。

- 親クラス: 4D.Entity
- クラス名: `DataClassNameEntity` (`DataClassName` はテーブル名です)
- 例: `cs.CityEntity`

計算属性

Entity クラスでは、専用のキーワードを使用して 計算属性 を定義することができます:

- `Function get attributeName`
- `Function set attributeName`
- `Function query attributeName`
- `Function orderBy attributeName`

詳細については、[計算属性](#) を参照してください。

エイリアス属性

Entity クラスでは、`Alias` キーワードを使用して エイリアス属性 を定義することができます (通常はリレート属性を対象に定義します):

```
Alias attributeName targetPath
```

詳細については、[エイリアス属性](#) を参照してください。

例題

```
// cs.CityEntity クラス

Class extends Entity

Function getPopulation()
    $0:=This.zips.sum("population")

Function isBigCity(): Boolean
// 関数 getPopulation() をクラス内で使用することができます
$0:=This.getPopulation()>50000
```

次のように関数を呼び出すことができます:

```

var $cityManager; $city : Object

$cityManager:=Open datastore(New object("hostname";"127.0.0.1:8111");"CityManager")
$city:=$cityManager.City.getCity("Caguas")

If ($city.isBigCity())
    ALERT($city.name + " は大きな町です。")
End if

```

定義規則

データモデルクラスを作成・編集する際には次のルールに留意しなくてはなりません:

- 4D のテーブル名は、cs **クラスストア** 内において自動的に DataClass クラス名として使用されるため、cs 名前空間において衝突があつてはなりません。特に:
 - 4D テーブルと **ユーザークラス名** に同じ名前を使用してはいけません。衝突が起きた場合には、ユーザークラスのコンストラクターは使用不可となります (コンパイラにより警告が返されます)。
 - 4D テーブルに予約語を使用してはいけません (例: "DataClass")。
- クラス定義の際、**Class extends** ステートメントに使用する親クラスの名前は完全に合致するものでなくてはいけません (文字の大小が区別されます)。たとえば、EntitySelection クラスを継承するには **Class extends EntitySelection** と書きます。
- データモデルクラスオブジェクトのインスタンス化に **new()** キーワードは使えません (エラーが返されます)。上述の ORDA クラステーブルに一覧化されている、通常の **インスタンス化の方法** を使う必要があります。
- 4D **クラスストア** のネイティブな ORDA クラス関数を、データモデルユーザークラス関数でオーバーライドすることはできません。

プリエンプティブ実行

コンパイル済みの状態では、データモデルクラス関数は次のように実行されます:

- シングルユーザー-application では、プリエンプティブまたはコオペラティブプロセスで実行されます (呼び出し元のプロセスに依存します)。
- クライアント/サーバー-application では、プリエンプティブプロセスで実行されます (ただし、**local** キーワードが使用されている場合は、シングルユーザーの場合と同様に、呼び出し元プロセスに依存します)。

クライアント/サーバーで動作するように設計されているプロジェクトでは、データモデルクラス関数のコードがスレッドセーフであることを確認してください。スレッドセーフでないコードが呼び出された場合、実行時にエラーが発生します (シングルユーザー-application ではコオペラティブ実行がサポートされているため、コンパイル時にはエラーが発生しません)。

計算属性

概要

計算属性は、計算をマスクするデータ型を持つデータクラス属性です。標準的な **4Dクラス** は、**get** (ゲッター) および **set** (セッター) **アクセサー関数** を用いて、計算プロパティの概念を実装しています。ORDA のデータクラス属性はこれを利用し、さらに **query** と **orderBy** の2つの関数で機能を拡張しています。

計算属性には最低限、その値がどのように算出されるかを記述した **get** 関数が必要です。属性にゲッター関数が定義されている場合、4D は対応するストレージスペースをデータストアに作成せず、代わりに属性がアクセスされるたびに関数のコードを実行します。属性がアクセスされなければ、コードも実行されません。

計算属性は、その属性に値が割り当てられたときに実行される **set** 関数を実装することもできます。セッター関数は、割り当てられた値をどのように処理するかを記述します。通常は、1つ以上のストレージ属性や、場合によっては他のエンティティにリダイレクトします。

ストレージ属性と同様に、計算属性も クエリ に含めることができます。デフォルトでは、ORDA のクエリで計算属性が使用された場合、その属性はエンティティ毎に一度計算されます。場合によっては、これで十分です。しかし、特にクライアント/サーバーにおいてはパフォーマンスを向上させるため、実際のデータクラス属性に基づいた **query** 関数を計算属性に実装することで、それらのインデックスの恩恵を受けることができます。

同様に、計算属性を **並べ替え** に含めることもできます。デフォルトでは、ORDA の並べ替えで計算属性が使用された場合、その属性はエンティティ毎に一度計算されます。クエリと同様に、実際のデータクラス属性に基づいた **orderBy** 関数を計算属性に実装することで、パフォーマンスを向上させる

ことができます。

計算属性の定義

計算属性を作成するには、データクラスの `Entity クラス` に `get` アクセサー関数を定義します。計算属性は、データクラス属性およびエンティティ属性として自動的に利用可能になります。

その他の計算属性の関数 (`set`、`query`、`orderBy`) も、Entityクラスに定義することができます。これらの関数の定義は任意です。

リレーションに基づくエイリアス属性は、そのターゲット属性のパスを格納する専用の `path` プロパティを持ちます。同じデータクラスの属性に基づくエイリアス属性は、ターゲット属性と同じプロパティを持ちます (`path` プロパティはありません)。

ORDA の計算属性は、デフォルトでは `公開` されません。計算属性を公開するには、`get` 関数 の定義に `exposed` キーワードを追加します。

`get` および `set` 関数は、クライアント/サーバー処理を最適化するために、`local` プロパティを持つこともできます。

Function `get <attributeName>`

シンタックス

```
{local} {exposed} Function get <attributeName>({$event : Object}) -> $result : type  
// コード
```

ゲッター 関数は、`attributeName` 計算属性を宣言するために必須です。`attributeName` がアクセスされるたびに、4D は `Function get` のコードを評価し、`$result` 値を返します。

計算属性は、他の計算属性の値を使用することができます。再帰的な呼び出しはエラーになります。

ゲッター 関数は、`$result` パラメーターに基づいて、計算属性のデータ型を定義します。以下の結果の型が可能です:

- スカラー (テキスト、ブール、日付、時間、数値)
- オブジェクト
- ピクチャー
- BLOB
- エンティティ (例: `cs.EmployeeEntity`)
- エンティティセレクション (例: `cs.EmployeeSelection`)

`$event` パラメーターは、以下のプロパティが含みます:

プロパティ	タイプ	説明
<code>attributeName</code>	テキスト	計算属性の名称
<code>dataClassName</code>	テキスト	データクラスの名称
<code>kind</code>	テキスト	"get"
<code>result</code>	バリアント	任意。スカラー属性が Null を返すようにするには、このプロパティを Null 値で追加します。

例題

- `fullName` 計算属性:

```

Function get fullName($event : Object)-> $fullName : Text

Case of
  : (This.firstName=Null) & (This.lastName=Null)
    $event.result:=Null // Null値を返すには result を使用します
  : (This.firstName=Null)
    $fullName:=This.lastName
  : (This.lastName=Null)
    $fullName:=This.firstName
Else
  $fullName:=This.firstName+" "+This.lastName
End case

```

- 計算属性は、エンティティにリレートされた属性に基づいて定義することができます。

```

Function get bigBoss($event : Object)-> $result: cs.EmployeeEntity
  $result:=This.manager.manager

```

- 計算属性は、エンティティセレクションにリレートされた属性に基づいて定義することができます。

```

Function get coWorkers($event : Object)-> $result: cs.EmployeeSelection
  If (This.manager.manager=Null)
    $result:=ds.Employee.newSelection()
  Else
    $result:=This.manager.directReports.minus(this)
  End if

```

Function set <attributeName>

シンタックス

```

{local} Function set <attributeName>($value : type {; $event : Object})
// コード

```

セッター 関数は、属性に値が割り当てられたときに実行されます。この関数は通常、入力値を処理し、その結果を 1つ以上の他の属性に転送します。

\$value パラメーターは、属性に割り当てられた値を受け取ります。

\$event パラメーターは、以下のプロパティが含みます:

プロパティ	タイプ	説明
attributeName	テキスト	計算属性の名称
dataClassName	テキスト	データクラスの名称
kind	テキスト	"set"
value	バリант	計算属性によって処理されるべき値

例題

```

Function set fullName($value : Text; $event : Object)
  var $p : Integer
  $p:=Position(" "; $value)
  This.firstname:=Substring($value; 1; $p-1) // "" if $p<0
  This.lastname:=Substring($value; $p+1)

```

Function query <attributeName>

シンタックス

```

Function query <attributeName>($event : Object)
Function query <attributeName>($event : Object) -> $result : Text
Function query <attributeName>($event : Object) -> $result : Object
// コード

```

このメソッドは 3種類のシンタックスを受け入れます:

- 最初のシンタックスでは、`$event.result` オブジェクトプロパティを通じてクエリ全体を処理します。
- 2番目と 3番目のシンタックスでは、関数は `$result` に値を返します:
 - `$result` がテキストの場合、それは有効なクエリ文字列でなければなりません。
 - `$result` がオブジェクトの場合、次の 2つのプロパティを含まなければなりません:

プロパティ	タイプ	説明
<code>\$result.query</code>	テキスト	プレースホルダー (:1, :2, など) を使った有効なクエリ文字列
<code>\$result.parameters</code>	コレクション	プレースホルダーに渡す値

`query` 関数は、計算属性を使用するクエリが開始されるたびに実行されます。インデックス付きの属性を利用することで、クエリをカスタマイズしたり最適化したりすることができます。計算属性に対して `query` 関数が実装されていない場合、検索は常にシーケンシャルにおこなわれます（`get <AttributeName>` 関数によるすべての値の評価に基づきます）。

以下の機能はサポートされていません:

- エンティティ、またはエンティティセレクション型の計算属性に対する `query` 関数の呼び出し
- 結果のクエリ文字列における `order by` キーワードの使用

`$event` パラメーターは、以下のプロパティが含みます:

プロパティ	タイプ	説明
attributeName	テキスト	計算属性の名称
dataClassName	テキスト	データクラスの名称
kind	テキスト	"query"
value	バリアント	計算属性によって処理されるべき値
operator	テキスト	クエリ演算子 (query クラス関数も参照ください)。とりうる値: <ul style="list-style-type: none"> ● == (と等しい; @ はワイルドカード) ● === (と等しい; @ はワイルドカードでない) ● != (と等しくない; @ はワイルドカード) ● !== (と等しくない; @ はワイルドカードでない) ● < (小さい) ● <= (less than or equal to) ● > (大きい) ● >= (以上) ● IN (含まれる) ● % (キーワードを含む)
result	バリアント	計算属性によって処理されるべき値。4D がデフォルトクエリ (計算属性では常にシーケンシャル) を実行するよう にしたい場合は、このプロパティに <code>Null</code> を渡します。

関数が `$result` に値を返し、`$event.result` プロパティにも別の値が割り当てられている場合、`$event.result` が優先されます。

例題

- `fullName` 計算属性のクエリ:

```

Function query fullName($event : Object)->$result : Object

    var $fullname; $firstname; $lastname; $query : Text
    var $operator : Text
    var $p : Integer
    var $parameters : Collection

    $operator:=$event.operator
    $fullname:=$event.value

    $p:=Position(" "; $fullname)
    If ($p>0)
        $firstname:=Substring($fullname; 1; $p-1)+"@"
        $lastname:=Substring($fullname; $p+1)+"@"
        $parameters:=New collection($firstname; $lastname) // 2要素のコレクション
    Else
        $fullname:=$fullname+"@"
        $parameters:=New collection($fullname) // 1要素のコレクション
    End if

    Case of
    : ($operator=="==") | ($operator=="===")
        If ($p>0)
            $query:="(firstName = :1 and lastName = :2) or (firstName = :2 and lastName = :1)"
        Else
            $query:="firstName = :1 or lastName = :1"
        End if
    : ($operator!="!=")
        If ($p>0)
            $query:="firstName != :1 and lastName != :2 and firstName != :2 and lastName != :1"
        Else
            $query:="firstName != :1 and lastName != :1"
        End if
    End case

    $result:=New object("query"; $query; "parameters"; $parameters)

```

ユーザーのテキスト入力に基づくクエリでは、セキュリティ上の理由からプレースホルダーを使用することが推奨されています（[query\(\)](#) の説明参照）。

呼び出しコードの例：

```
$emps:=ds.Employee.query("fullName = :1"; "Flora Pionsin")
```

- この関数は *age* (年齢) 計算属性に対するクエリを処理し、パラメーターを含むオブジェクトを返します：

```

Function query age($event : Object)->$result : Object

    var $operator : Text
    var $age : Integer
    var $_ages : Collection

    $operator:=$event.operator

    $age:=Num($event.value) // 整数
    $d1:=Add to date( Current date; -$age-1; 0; 0)
    $d2:=Add to date($d1; 1; 0; 0)
    $parameters:=New collection($d1; $d2)

Case of

    : ($operator=="==")
        $query:="birthday > :1 and birthday <= :2" // d1 より大きい、かつ d2 以下

    : ($operator=="===")
        $query:="birthday = :2" // d2 = 2つ目の算出値 (= 誕生日)

    : ($operator==">=")
        $query:="birthday <= :2"

    //... その他の演算子

End case

If (Undefined($event.result))
    $result:=New object
    $result.query:=$query
    $result.parameters:=$parameters
End if

```

呼び出しコードの例:

```

// 20歳以上で 21歳未満の人
$twenty:=people.query("age = 20") // "==" のケースを呼び出します

// 本日満 20歳になった人
$twentyToday:=people.query("age === 20") // people.query("age is 20") と同じ

```

Function orderBy <attributeName>

シンタックス

```

Function orderBy <attributeName>($event : Object)
Function orderBy <attributeName>($event : Object)-> $result : Text

// コード

```

`orderBy` 関数は、計算属性で並べ替えられるたびに実行されます。これにより、計算属性で並べ替えることができます。たとえば、`fullName` を名字、名前の順にソートしたり、逆に名字、名前の順にソートすることができます。計算属性に対して `orderBy` 関数が実装されていない場合、並べ替えは常にシーケンシャルにおこなわれます（`get <AttributeName>` 関数によるすべての値の評価に基づきます）。

Entity クラス、または EntitySelection クラス型の計算属性に対する `orderBy` 関数の呼び出しはサポートされていません。

`$event` パラメーターは、以下のプロパティが含みます：

プロパティ	タイプ	説明
<code>attributeName</code>	テキスト	計算属性の名称
<code>dataClassName</code>	テキスト	データクラスの名称
<code>kind</code>	テキスト	"orderBy"
<code>value</code>	バリアント	計算属性によって処理されるべき値
<code>operator</code>	テキスト	"desc" または "asc" (デフォルト)
<code>descending</code>	ブール	降順の場合は <code>true</code> , 昇順の場合は <code>false</code>
<code>result</code>	バリアント	計算属性によって処理されるべき値。4D にデフォルツートを実行させるには、 <code>Null</code> を渡します。

`operator` と `descending` プロパティのどちらを使っても構いません。これは、基本的にプログラミングのスタイルの問題です (例題参照)。

`orderBy` 文字列は、`$event.result` オブジェクトプロパティまたは関数の戻り値である `$result` のどちらにでも返すことができます。関数が `$result` に値を返し、`$event.result` プロパティにも別の値が割り当てられている場合、`$event.result` が優先されます。

例題

次のような条件分岐のコードを書くことができます：

```
Function orderBy fullName($event : Object)-> $result : Text
    If ($event.descending=True)
        $result:="firstName desc, lastName desc"
    Else
        $result:="firstName, lastName"
    End if
```

また、次のような短縮コードを書くこともできます：

```
Function orderBy fullName($event : Object)-> $result : Text
    $result:="firstName "+$event.operator+", lastName "+$event.operator
```

場合によっては条件分岐のコードが必要です：

```
Function orderBy age($event : Object)-> $result : Text
    If ($event.descending=True)
        $result:="birthday asc"
    Else
        $result:="birthday desc"
    End if
```

エイリアス属性

概要

エイリアス属性は、ターゲット 属性と呼ばれるデータモデルの別の属性を元に定義されます。ターゲット属性には、リレートデータクラス (リレートレベルは

無制限) または同じデータクラスのものを使用できます。エイリアス属性はデータではなく、ターゲット属性へのパスを格納します。データクラスには、必要な数だけエイリアス属性を定義することができます。

エイリアス属性は、N対Nリレーションを扱うのに便利です。実装の詳細ではなくビジネスの概念を扱ってコードやクエリを作成できるため、これらの可読性が向上します。

エイリアス属性の定義

データクラス内にエイリアス属性を作成するには、データクラスの [Entityクラス](#) において `Alias` キーワードを使用します。

```
Alias <attributeName> <targetPath>
```

シンタックス

```
{exposed} Alias <attributeName> <targetPath>
```

`attributeName` は、[プロパティ名の命名規則](#) に準拠している必要があります。

`targetPath` は、"employee.company.name" のような、1つ以上のレベルを含む属性パスです。ターゲット属性が同じデータクラスに属している場合、`targetPath` は属性名となります。

エイリアスは、他のエイリアスのパスに使用することができます。

[計算属性](#) もエイリアスパスに使用することができますが、パスの最後のレベルとしてのみ使用できます。そうでない場合は、エラーが返されます。たとえば、"fullName" 計算属性がある場合、"employee.fullName" というエイリアスパスは有効です。

ORDA のエイリアス属性は、デフォルトでは 公開されません。リモートリクエストでエイリアスを利用するには、`Alias` キーワードの前に `exposed` キーワードを追加する必要があります。

エイリアス属性の使用

エイリアス属性は読み取り専用です (同じデータクラスのスカラー属性に基づく場合は例外です；最後の例題参照)。エイリアス属性は、次のようなクラス関数において、ターゲット属性パスの代わりに使用することができます：

Function

```
dataClass.query() , entitySelection.query()  
entity.toObject()  
entitySelection.toCollection()  
entitySelection.extract()  
entitySelection.orderBy()  
entitySelection.orderByFormula()  
entitySelection.average()  
entitySelection.count()  
entitySelection.distinct()  
entitySelection.sum()  
entitySelection.min()  
entitySelection.max()  
entity.diff()  
entity.touchedAttributes()
```

エイリアス属性はサーバー上で計算されることに留意してください。リモート環境において、エンティティのエイリアス属性を更新するには、エンティティをサーバーから再ロードする必要があります。

エイリアスのプロパティ

エイリアス属性の `kind` プロパティ (属性の種類) は "alias" です。

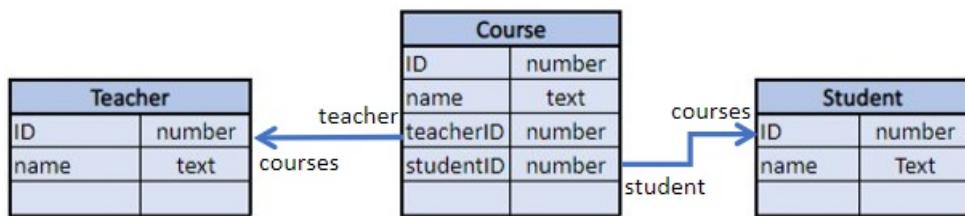
エイリアス属性は、ターゲット属性の `type` プロパティを継承します。

- ターゲット属性の `kind` プロパティが "storage" の場合、エイリアス属性の `type` はターゲット属性と同じになります。
- ターゲット属性の `kind` が "relatedEntity" または "relatedEntities" の場合、エイリアスの `type` は `4D.Entity` または `4D.EntitySelection ("classnameEntity" または "classnameSelection")` になります。

リレーションに基づくエイリアス属性は、そのターゲット属性のパスを格納する専用の `path` プロパティを持ちます。同じデータクラスの属性に基づくエイリアス属性は、ターゲット属性と同じプロパティを持ちます (`path` プロパティはありません)。

例題

以下のモデルがあるとき:



Teacher データクラスに、教師の生徒をすべて返すエイリアス属性を定義します:

```
// cs.TeacherEntity クラス
Class extends Entity
Alias students courses.student //relatedEntities
```

Student データクラスには、生徒の教師をすべて返すエイリアス属性を定義します:

```
// cs.StudentEntity クラス
Class extends Entity
Alias teachers courses.teacher //relatedEntities
```

Course データクラスには次を定義します:

- "name" 属性を別名で参照するためのエイリアス属性
- 教師の名前を返すエイリアス属性
- 生徒の名前を返すエイリアス属性

```
// cs.CourseEntity クラス
Class extends Entity
Exposed Alias courseName name //スカラー値
Exposed Alias teacherName teacher.name //スカラー値
Exposed Alias studentName student.name //スカラー値
```

すると、以下のクエリを実行することができます：

```
// "Archaeology" の授業を検索します
ds.Course.query("courseName = :1";"Archaeology")

// Smith 教師が教えている授業を検索します
ds.Course.query("teacherName = :1";"Smith")

// 生徒 "Martin" が参加している授業を検索します
ds.Course.query("studentName = :1";"Martin")

// M. Smith 教師の生徒を検索します
ds.Student.query("teachers.name = :1";"Smith")

// M. Martin を生徒を持つ教師を検索します
ds.Teacher.query("students.name = :1";"Martin")
// シンプルなクエリ文字列で複雑なクエリを実行していることに注目してください
// queryPlan は次のとおりです：
// "Join on Table : Course : Teacher.ID = Course.teacherID,
// subquery:[ Join on Table : Student : Course.studentID = Student.ID,
// subquery:[ Student.name === Martin]]"
```

courseName エイリアスの値は編集することができます：

```
// エイリアス属性を使って、授業の名称を変更します
$arch:=ds.Course.query("courseName = :1";"Archaeology")
$arch.courseName:="Archaeology II"
$arch.save() //courseName と name は "Archaeology II" に変更されます
```

公開vs非公開関数

セキュリティ上の理由により、データモデルクラス関数およびエイリアス属性はデフォルトですべて、リモートリクエストに対し 非公開（つまりプライベート）に設定されています。

リモートリクエストには次のものが含まれます：

- `Open datastore` によって接続されたリモートの 4D アプリケーションが送信するリクエスト
- REST リクエスト

通常の 4D クライアント/サーバーリクエストは影響されません。このアーキテクチャにおいては、データモデルクラス関数は常に利用可能です。

公開されていない関数はリモートアプリケーションで利用することができず、REST リクエストによるオブジェクトインスタンスに対して呼び出すこともできません。リモートアプリケーションが非公開関数をアクセスしようとすると、"-10729 (未知のメンバー機能です)" エラーが返されます。

リモートリクエストによる呼び出しを許可するには、`exposed` キーワードを使ってデータモデルクラス関数を明示的に宣言する必要があります。シンタックスは次の通りです：

```
// 公開関数の宣言
exposed Function <functionName>
```

`exposed` キーワードは、データモデルクラス関数に対してのみ利用可能です。[通常のユーザークラス](#) 関数に対して使った場合、キーワードは無視され、コンパイラはエラーを返します。

例題

公開された関数によって、DataClass クラスのプライベート関数を呼び出します：

```

Class extends DataClass

// 公開関数
exposed Function registerNewStudent($student : Object) -> $status : Object

var $entity : cs.StudentsEntity

$entity:=ds.Students.new()
$entity.fromObject($student)
$entity.school:=This.query("name=:1"; $student.schoolName).first()
$entity.serialNumber:=This.computeSerialNumber()
$status:=$entity.save()

// 非公開（プライベート）関数
Function computeIDNumber()-> $id : Integer
// 新規ID番号を算出します
$id:=...

```

呼び出し元のコードは次の通りです：

```

var $remoteDS; $student; $status : Object
var $id : Integer

$remoteDS:=Open datastore(New object("hostname"; "127.0.0.1:8044"); "students")
$student:=New object("firstname"; "Mary"; "lastname"; "Smith"; "schoolName"; "Math school")

$status:=$remoteDS.Schools.registerNewStudent($student) // OK
$id:=$remoteDS.Schools.computeIDNumber() // エラー（未知のメンバー機能です）

```

ローカル関数

クライアント/サーバーアーキテクチャではデフォルトで、ORDA データモデル関数は サーバー上で 実行されます。関数リクエストとその結果だけが通信されるため、通常はベストパフォーマンスが提供されます。

しかしながら、状況によってはその関数はクライアント側で完結するものかもしれません（たとえば、すでにローカルキャッシュにあるデータを処理する場合など）。そのような場合には、`local` キーワードを使ってサーバーへのリクエストをおこなわないようにし、アプリケーションのパフォーマンスを向上させることができます。シンタックスは次の通りです：

```

// クライアント/サーバーにおいてローカル実行する関数の宣言
local Function <functionName>

```

このキーワードを使うと、関数は常にクライアントサイドで実行されます。

`local` キーワードは、データモデルクラス関数に対してのみ利用可能です。[通常のユーザークラス](#) 関数に対して使った場合、キーワードは無視され、コンパイラはエラーを返します。

最終的にサーバーへのアクセスが必要になっても（ORDAキャッシュが有効期限切れになった場合など）関数は動作します。もっとも、それではローカル実行によるパフォーマンスの向上は見込めないため、ローカル関数がサーバー上のデータにアクセスしないことを確認しておくことが推奨されます。サーバーに対して複数のリクエストをおこなうローカル関数は、サーバー上で実行されて結果だけを返す関数よりも非効率的です。たとえば、Schools Entityクラスの次の関数を考えます：

```
// 2000年以降の生まれの生徒を検索します
// local キーワードを適切に使用していない例です
local Function getYoungest
    var $0 : Object
    $0:=This.students.query("birthDate >= :1"; !2000-01-01!).orderBy("birthDate desc").slice(0; 5)
```

- `local` キーワードを使わない場合、1つのリクエストで結果が得られます。
- `local` キーワードを使う場合、4つのリクエストが必要になります：Schools エンティティの `students` エンティティセレクションの取得、`query()` の実行、`orderBy()` の実行、`slice()` の実行。この例では、`local` キーワードを使用するのは適切ではありません。

例題

年齢の計算

`birthDate` (生年月日) 属性を持つエンティティがある場合に、リストボックス内で呼び出すための `age()` 関数を定義します。この関数をクライアントサイドで実行することで、リストボックスの各行がサーバーへのリクエストを生成するのを防ぎます。

`StudentsEntity` クラス：

```
Class extends Entity

local Function age() -> $age: Variant

If (This.birthDate#!00-00-00!)
    $age:=Year of(Current date)-Year of(This.birthDate)
Else
    $age:=Null
End if
```

属性のチェック

クライアントにロードされ、ユーザーによって更新されたエンティティの属性について、サーバーへ保存リクエストを出すまえに、それらの一貫性を検査します。

`StudentsEntity` クラスのローカル関数 `checkData()` は生徒の年齢をチェックします：

```
Class extends Entity

local Function checkData() -> $status : Object

$status:=New object("success"; True)
Case of
    : (This.age()=Null)
        $status.success:=False
        $status.statusText:="生年月日が入力されていません。"

    :((This.age() <15) | (This.age()>30) )
        $status.success:=False
        $status.statusText:="生徒の年齢は 15 ~ 30 の範囲で入力してください。この生徒の年齢は "+String(This.age())+
End case
```

呼び出し元のコード：

```

var $status : Object

// Form.student は全属性とともにロードされており、フォーム上で更新されました
$status:=Form.student.checkData()
If ($status.success)
    $status:=Form.student.save() // サーバーを呼び出します
End if

```

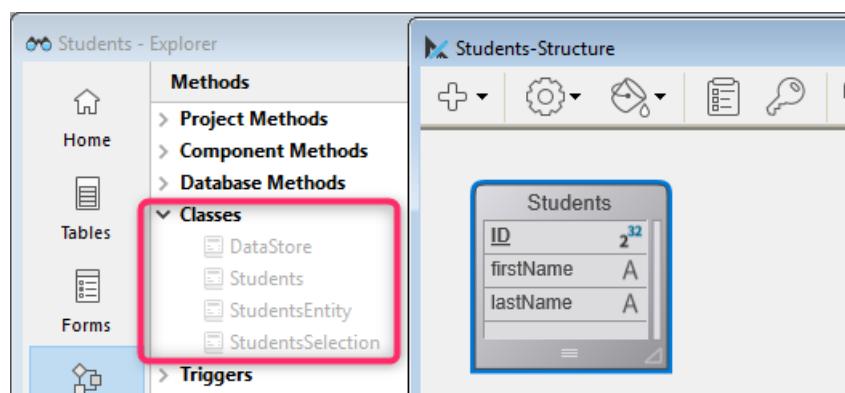
4D IDE (統合開発環境) におけるサポート

クラスファイル

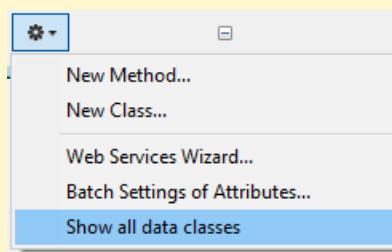
ORDA データモデルユーザークラスは、クラスと同じ名称の .4dm ファイルを **通常のクラスファイルと同じ場所** (つまり、Project フォルダー内の /Sources/Classes フォルダー) に追加することで定義されます。たとえば、Utilities データクラスのエンティティクラスは、UtilitiesEntity.4dm ファイルによって定義されます。

クラスの作成

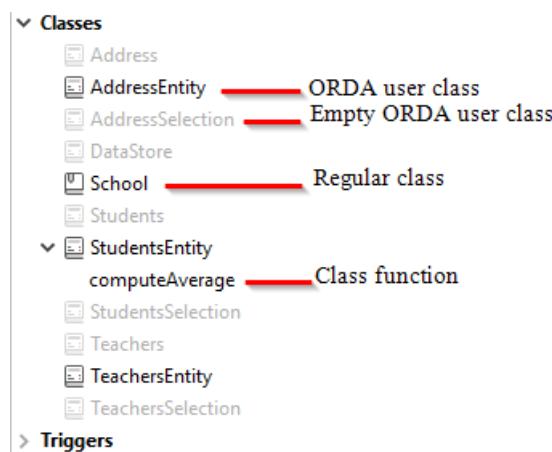
各データモデルオブジェクトに関わるクラスは、4D によってあらかじめ自動的にメモリ内に作成されます。



空の ORDA クラスは、デフォルトではエクスプローラーに表示されません。表示するにはエクスプローラーのオプションメニューより データクラスを全て表示 を選択します:



ORDA ユーザークラスは通常のクラスとは異なるアイコンで表されます。空のクラスは薄く表示されます:



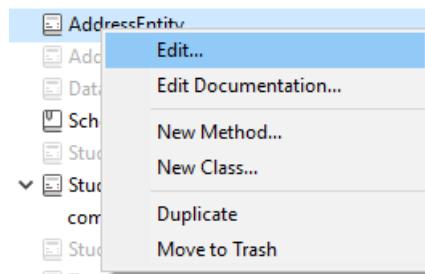
ORDA クラスファイルを作成するには、エクスプローラーで任意のクラスをダブルクリックします。4D はクラスファイルを作成し、`extends` ステートメントを自動で追加します。たとえば、Entity クラスを継承するクラスの場合は:

Class extends Entity

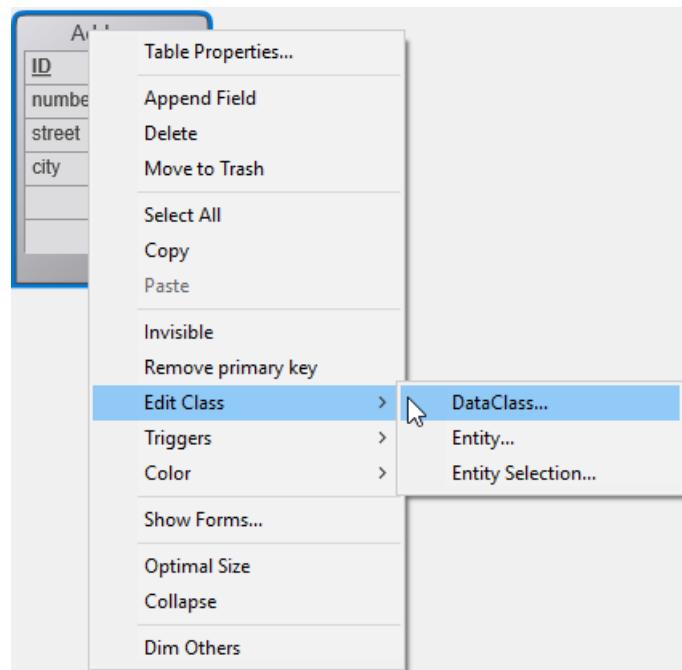
定義されたクラスはエクスプローラー内で濃く表示されます。

クラスの編集

定義された ORDA クラスファイルを 4D メソッドエディターで開くには、ORDA クラス名を選択してエクスプローラーのオプションメニュー、またはコンテキストメニューの **編集...** を使用するか、ORDA クラス名をダブルクリックします:



ローカルデータストア (`ds`) に基づいた ORDA クラスの場合には、4D ストラクチャーウィンドウからも直接クラスコードにアクセスできます:



メソッドエディター

4D メソッドエディターにおいて、ORDA クラス型として定義された変数は、自動補完機能の対象となります。Entity クラス変数の例です:

```
var $student : cs.StudentsEntity
$student:=ds.Students.all().first()
$student.|
```

A screenshot of the 4D Method Editor showing code completion for a variable '\$student'. The variable is typed as '\$student.' followed by a period. A dropdown menu shows various properties and methods: 'ID', 'name', 'clone', 'computeAverage' (which is highlighted with a red rectangle), 'diff', 'drop', 'first', and 'fromObject'. The 'computeAverage' method is likely a custom method defined in the Entity class.

データ操作

ORDA では、[エンティティ](#) および [エンティティセレクション](#) を介してデータにアクセスします。これらのオブジェクトを使って、データストアのデータを作成・更新・クエリ・ソートすることができます。

エンティティの作成

データクラス内に新しいエンティティを作成する方法は二つあります:

- エンティティはデータベースレコードへの参照であるため、"クラシックな" 4Dランゲージを使用してレコードを作成し、それを `entity.next()` や `entitySelection.first()` といった ORDAメソッドで参照することでエンティティを作成できます。
- また、`dataClass.new()` メソッドを使用することでもエンティティも作成することができます。

エンティティはメモリ内にしか作成されないという点に注意してください。データストアに追加したい場合、`entity.save()` メソッドを呼ぶ必要があります。

エンティティ属性は、エンティティオブジェクトのプロパティとして直接利用可能です。詳細な情報については、[エンティティ属性の使用](#) を参照してください。

たとえば、カレントデータストア内の "Employee" データクラスに新しいエンティティを作成し、`firstname` と `name` 属性に "John" と "Dupont" を割り当てる場合を考えます:

```
var $myEntity : cs.EmployeeEntity
$myEntity:=ds.Employee.new() // エンティティ型の新規オブジェクトを作成します
$myEntity.name:="Dupont" // 'Dupont' を 'name' 属性に代入します
$myEntity.firstname:="John" // 'John' を 'firstname' 属性に代入します
$myEntity.save() // エンティティを保存します
```

エンティティは、それが作成されたプロセス内でのみ定義されます。そのため、たとえばエンティティへの参照を、インタープロセス変数内に保存して他のプロセスで使用する、といったことはできません。

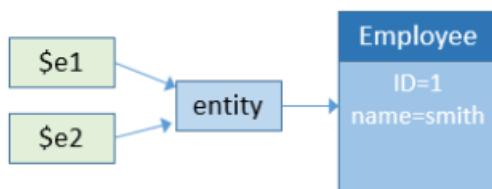
エンティティと参照

エンティティには、4Dレコードへの参照が格納されています。異なるエンティティが同じ 4Dレコードを参照することもあり得ます。また、エンティティは 4Dオブジェクト変数に保存可能であることから、異なる変数が同じエンティティへの参照を格納していることもあり得ます。

以下のコードを実行した場合:

```
var $e1; $e2 : cs.EmployeeEntity
$e1:=ds.Employee.get(1) // ID 1をもつ社員にアクセスします
$e2:=$e1
$e1.name:="Hammer"
// $e1 も $e2 も、どちらも同じエンティティへの参照を共有します
// $e2.name の中身も "Hammer" です
```

これは以下のように図解することができます:



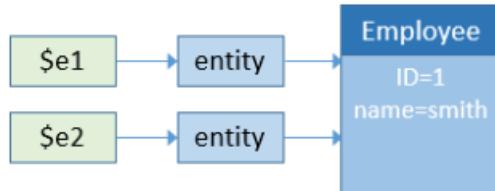
次に、以下のコードを実行した場合:

```

var $e1; $e2 : cs.EmployeeEntity
$e1:=ds.Employee.get(1)
$e2:=ds.Employee.get(1)
$e1.name:="Hammer"
//変数 $e1 はエンティティへの参照を格納しています
//変数 $e2 は別のエンティティへの参照を格納しています
//$e2.name の中身は "smith" です

```

これは以下のように図解することができます:



しかし、両方のエンティティが同じレコードを参照していることに注意してください。どちらの場合でも、`entity.save()` メソッドを呼び出した場合、レコードは更新されます（衝突が発生した場合を除きます。[エンティティロック](#) 参照）。

実際には、`$e1` も `$e2` もエンティティそのものではなく、エンティティへの参照です。これはつまり、どのような関数やメソッドにも直接受け渡すことができ、ポインターのように振る舞うということです。そしてこれは 4D ポインターよりもずっと高速です。たとえば:

```

For each($entity;$selection)
    do_Capitalize($entity)
End for each

```

そして `do_Capitalize` メソッドが以下のような形であった場合:

```

$entity:=$1
$name:=$entity.lastname
If(Not($name=NULL))
    $name:=Uppercase(Substring($name;1;1))+Lowercase(Substring($name;2))
End if
$entity.lastname:=$name

```

他の 4D のオブジェクトと同様にエンティティを扱うことができ、[引数](#) としてその参照を渡すことができます。

エンティティでは、クラシックな 4D 言語のような "カレントレコード" という概念はありません。エンティティは、いくつでも必要な数を同時に使用することができます。また、エンティティには自動ロックの機構が備わっています（[エンティティロック](#) 参照）。エンティティの読み込みには、[レイジーローディング](#) 機構が使用されます。これはつまり必要な分の情報だけが読み込まれるということです。いずれにせよ、クライアント/サーバーでは必要であればエンティティを直接自動的に読み込むことも可能です。

エンティティ属性の使用

エンティティ属性はデータを保存し、対応するテーブルの対応するフィールドをマップします。ストレージ型のエンティティ属性はエンティティオブジェクトの単純なプロパティとして設定や取得ができますが、リレートエンティティ (relatedEntity) 型とリレートエンティティズ (relatedEntities) 型のエンティティ属性はエンティティあるいはエンティティセレクションを返します。

属性の型についての詳細な情報については、[ストレージ属性とリレーション属性](#) の段落を参照してください。

たとえば、ストレージ属性を設定するためには:

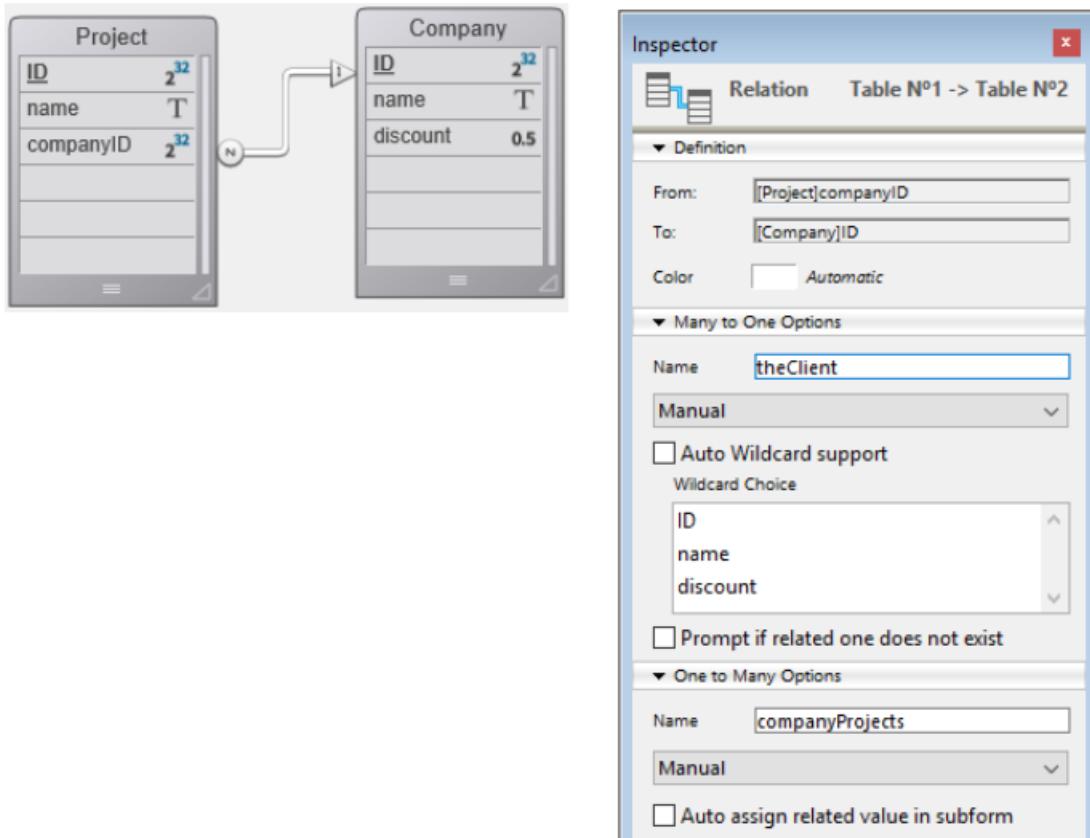
```

$entity:=ds.Employee.get(1) // ID1 の社員エンティティを取得します
$name:=$entity.lastname // 社員のラストネームを取得します。例: "Smith"
$entity.lastname:="Jones" // 社員のラストネームを変更します
$entity.save() // 変更を保存します

```

データベースの BLOBフィールド（スカラーBLOB）は、ORDAで扱われるにあたって、BLOBオブジェクト属性（[4D.Blob](#)）に自動変換されます。BLOBオブジェクト属性を保存する際には、（利用可能なメモリによってのみサイズ制限される BLOBオブジェクトとは異なり）BLOBフィールドのサイズが 2GB に制限されることに注意してください。

リレート属性にアクセスできるかどうかは、属性の型によります。たとえば、以下のようなストラクチャーがあるとき：



リレートされたオブジェクトを通してデータにアクセスすることができます：

```

$entity:=ds.Project.all().first().theClient // 先頭プロジェクトに関連する Company エンティティを取得します
$EntitySel:=ds.Company.all().first().companyProjects // 先頭の会社に関連する Project エンティティセレクションを

```

上記の例において、*theClient* と *companyProjects* はどちらもプライマリーリレーション属性であり、二つのデータクラス間の直接的なリレーションを表すことに注意してください。しかしながら、複数のレベルのリレーションを通したパスに基づいてリレーション属性をビルドすることも可能です（循環参照含む）。たとえば、以下のようなストラクチャーの場合を考えます：

Inspector

Relation Table N°1 -> Table N°1

Definition

From: [Employee]managerID
To: [Employee]ID
Color: Automatic

Many to One Options

Name: manager
Manual
 Auto Wildcard support
Wildcard Choice:
ID
firstname
lastname
 Prompt if related one does not exist

One to Many Options

Name: directReports
Manual
 Auto assign related value in subform
▶ Deletion Control
▶ SQL

社員はそれぞれマネージャーにもなりえますし、マネージャーを持つこともできます。ある社員のマネージャーのマネージャーを取得したい場合、以下のように書くことができます：

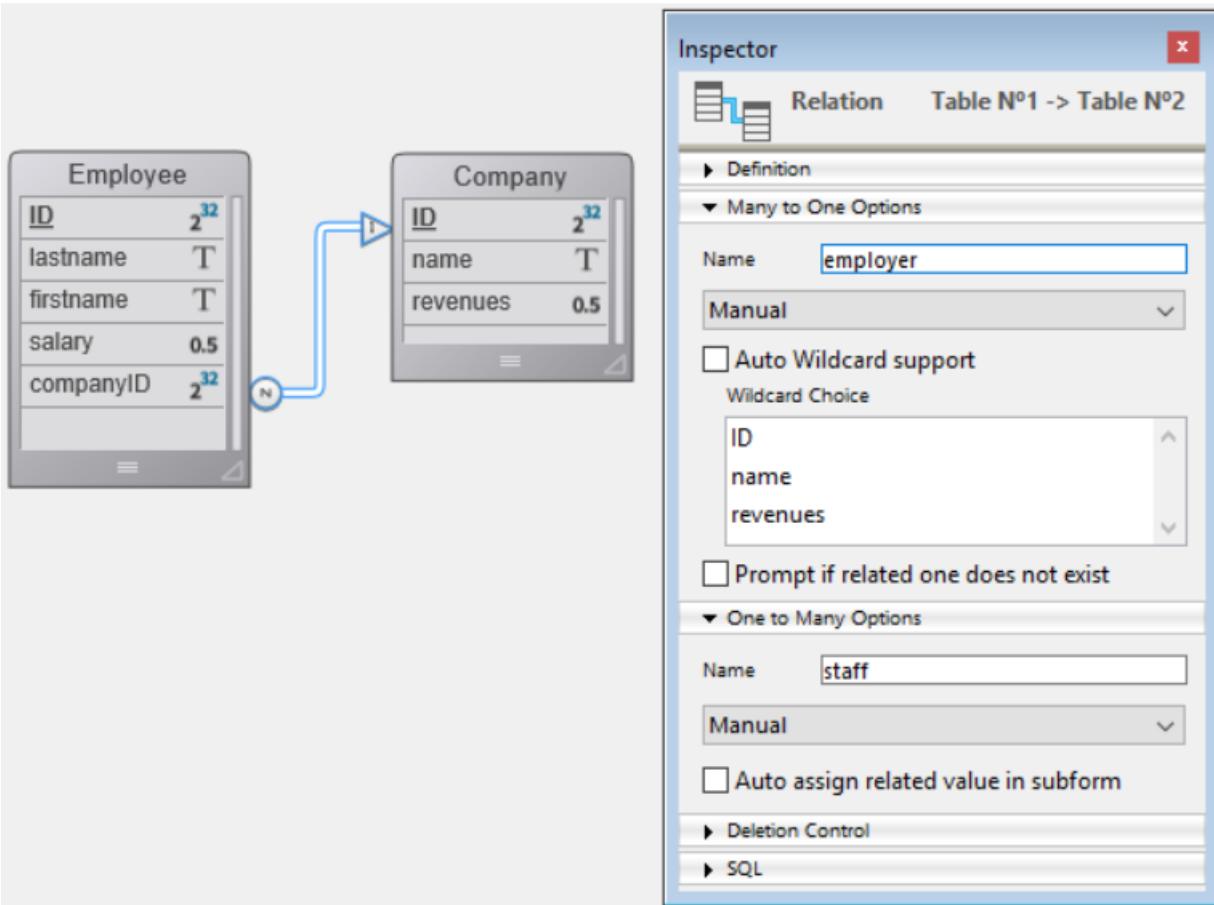
```
$myEmp:=ds.Employee.get(50)
$manLev2:=$myEmp.manager.manager.lastname
```

リレーション属性への値の代入

ORDAアーキテクチャーでは、リレーション属性はエンティティにリレートされたデータを直接格納します：

- N対1型リレーション属性 (リレートエンティティ (relatedEntity) 型) は一つのエンティティを格納します。
- 1対N型リレーション属性 (リレートエンティティズ (relatedEntities) 型) はエンティティセレクションを格納します。

以下の (単純化された) ストラクチャーを見てみましょう：



この例では、"Employee" データクラスに属するエンティティの "employer" 属性には、エンティティ型のオブジェクト (あるいは null 値) が格納されます。"Company" データクラスに属するエンティティの "staff" 属性には、エンティティセレクション型のオブジェクト (あるいは null 値) が格納されます。

ORDAでは、リレーションの自動あるいは手動プロパティは何の効力も持ちません。

"employer" 属性に直接値を代入したい場合には、"Company" データクラスの既存エンティティを渡す必要があります。たとえば:

```
$emp:=ds.Employee.new() // 新規の社員エンティティを作成します
$emp.lastname:="Smith" // 属性に値を代入します
$emp.employer:=ds.Company.query("name =:1";"4D")[0] // リレーション属性に会社エンティティを代入します
$emp.save()
```

4D では、"1" エンティティにリレートされている N エンティティ側のリレーション属性への入力を容易にするための追加の手段が提供されています: リレーション属性に代入する際に、"1" エンティティのプライマリーキーを直接渡す方法です。これが動作するためには、数値あるいはテキスト型のデータ (プライマリーキー値) をリレーション属性に渡します。すると 4D はデータクラス内の該当するエンティティを自動的に検索してくれます。たとえば:

```
$emp:=ds.Employee.new()
$emp.lastname:="Wesson"
$emp.employer:=2 // リレーション属性にプライマリーキーを代入します
// 4D はプライマリーキー (この場合、ID) の値が 2 である会社を検索し、
// それを社員に代入します
$emp.save()
```

これはとくに、リレーションナルデータベースから大量のデータを読み込むときに有用です。このような読み込みでは通常 "ID" カラムが含まれており、これはリレーション属性に直接割り当て可能なプライマリーキーを参照しています。

これはまた、1 データクラス側で対応するエンティティを事前に作成することなく N エンティティ側のプライマリーキーを割り当てることができるということです。リレートされているデータクラスに存在しないプライマリーキーを割り当てた場合、それは保管され、"1" データクラス側でエンティティが作成されたときに 4D によって割り当てられます。

"N" データクラス側のリレーション属性を通して、"1" リレートエンティティの属性値を、直接代入・変更することができます。たとえば、Employee エンティティにリレートされている Company エンティティの name 属性を変更したい場合、以下のように書くことができます:

```

$emp:=ds.Employee.get(2) // プライマリーキーが 2 の Employee エンティティを読み込みます
$emp.employer.name:="4D, Inc." // リレートされている Company の name 属性を変更します
$emp.employer.save() // リレーション属性の変更を保存します
// リレートされているエンティティも更新されます

```

エンティティセレクションの作成

以下の方法で、[エンティティセレクション](#) 型のオブジェクトを作成することができます：

- データクラス または [既存のエンティティセレクション](#) のエンティティに対してクエリを実行する；
- [.all\(\)](#) DataClassクラス関数を使用して、データクラス内の全エンティティを選択する；
- [Create entity selection](#) コマンドあるいは [.newSelection\(\)](#) DataClassクラス関数を使用して空のエンティティコレクションオブジェクトを作成する；
- [.copy\(\)](#) EntitySelectionクラス関数を使用して、既存のエンティティセレクションを複製する；
- [EntitySelectionクラス](#) の様々な関数の中から、[.or\(\)](#) のように新しいエンティティセレクションを返すものを使用する；
- "リレートエンティティズ" 型のリレーション属性を使用する（以下参照）

データクラスに対して、異なるエンティティセレクションを好きなだけ同時に作成し、使用することができます。エンティティセレクションは、エンティティへの参照を格納しているに過ぎないという点に注意してください。異なるエンティティセレクションが同じエンティティへの参照を格納することも可能です。

共有可能/追加可能なエンティティセレクション

エンティティセレクションには 2種類あります：共有可能 (shareable)（複数のプロセスで読み込み可能、ただし追加不可）のものと、追加可能 (alterable)（[add\(\)](#) 関数が使用可能、ただしカレントプロセスでのみ利用可）のものです：

プロパティ

共有可能 なエンティティセレクションは以下の特徴を持ちます：

- 共有オブジェクトまたは共有コレクションに保存する事が可能で、複数のプロセス間あるいはワーカー間で引数として受け渡しすることができます。
- 複数の共有オブジェクトまたは共有コレクションに保存する事が可能です。また、グループに属している共有オブジェクトまたは共有コレクションに保存することも可能です（つまり、ロック識別子 を持っていないということです）。
- 新たにエンティティを追加することはできません。共有可能なエンティティセレクションに対してエンティティを追加しようとした場合、エラーがトリガーされます（エラー-1637 - このエンティティセレクションは編集不可です）。共有可能なエンティティセレクションに対してエンティティを追加したい場合、[.add\(\)](#) 関数を呼び出す前に、[.copy\(\)](#) 関数を使用して共有不可のエンティティセレクションへと変換する必要があります。

大多数のエンティティセレクション関数（[.slice\(\)](#), [.and\(\)](#) 等）は、呼び出し対象のエンティティセレクションを変更せずに新規のエンティティセレクションを返すため、共有可能なエンティティセレクションに対して使用できます。

追加可能 なエンティティセレクションは以下の特徴を持ちます：

- プロセス間での共有はできません。また共有オブジェクト/コレクションへの保存もできません。共有不可のエンティティセレクションを共有オブジェクト/コレクションに保存しようとした場合、エラーがトリガーされます（エラー -10721 - 共有オブジェクトまたはコレクションにおいてサポートされる値の型ではありません）。
- 新規エンティティを受け取ることができます（つまり、[.add\(\)](#) 関数を使用できます）。

共有可能/追加可能エンティティセレクションの定義

エンティティセレクションが 共有可能 または 追加可能 のいずれの特性を持つかは、そのエンティティセレクションの作成時に定義され、あとから変更することはできません。エンティティセレクションの特性は、[.isAlterable\(\)](#) 関数または [OB Is shared](#) コマンドを使って確認することができます。

新規のエンティティセレクションは次の場合に 共有可能 です：

- データクラスに対して呼び出された ORDAクラス関数によって生成された場合：[dataClass.all\(\)](#), [dataClass.fromCollection\(\)](#), [dataClass.query\(\)](#) 等。
- リレーション属性をもとに生成され、[entity.attributeName](#) (例: "company.employees") の attributeName が 1対Nリレーション属性で、かつ entity 自身がエンティティセレクションに属していない場合。
- [ck shared](#) オプションを指定したうえで、[entitySelection.copy\(\)](#) または [OB Copy](#) を使用し、明示的に共有可能としてコピーされた

場合。

例:

```
$myComp:=ds.Company.get(2) // $myComp はエンティティセレクションに属していません  
$employees:=$myComp.employees // $employees は共有可能です
```

新規のエンティティセレクションは次の場合に 追加可能 です:

- `dataClass.newSelection()` 関数または `Create entity selection` コマンドを使用して新規作成された空のエンティティセレクションの場合。
- `ck shared` オプションを指定せずに、`entitySelection.copy()` または `OB Copy` を使用し、明示的に追加可能としてコピーされた場合。

例:

```
$toModify:=ds.Company.all().copy() // $toModify は追加可能です
```

新規のエンティティセレクションは次の場合に、元となるエンティティセレクションの特性を 繙承 します:

- 既存のエンティティセレクションに対して呼び出された ORDAクラス関数 (`.query()`, `.slice()`, 等) によって生成された場合 .
- リレーションに基づいて生成された場合:
 - `entity.attributeName` (例: "company.employees") の `attributeName` が 1対Nリレーション属性で、かつ `entity` 自身がエンティティセレクションに属している場合 (`entity.getSelection()` エンティティセレクションと同じ特性になります)。
 - `entitySelection.attributeName` (例: "employees.employer") の `attributeName` がリレーション属性の場合 (エンティティセレクションと同じ特性になります)。
 - `entitySelection.extract()` から返されるコレクションがエンティティセレクションを含む場合 (エンティティセレクションと同じ特性になります)。

例:

```
$highSal:=ds.Employee.query("salary >= :1"; 1000000)  
    // データクラスに対するクエリによって生成されたため $highSal は共有可能です  
$comp:=$highSal.employer // $highSal が共有可能なため $comp も共有可能です  
  
$lowSal:=ds.Employee.query("salary <= :1"; 10000).copy()  
    // オプション無しの copy( ) によって生成されたため $lowSal は追加可能です  
$comp2:=$lowSal.employer // $lowSal が追加可能なため $comp2 も追加可能です
```

プロセス間のエンティティセレクションの共有 (例題)

二つのエンティティセレクションを使用し、それらをワーカープロセスに渡して適切な相手にメールを送信したい場合を考えます:

```
var $paid; $unpaid : cs.InvoicesSelection  
// 支払済および未払いの請求書のエンティティセレクションをそれぞれ取得します  
$paid:=ds.Invoices.query("status=:1"; "支払済")  
$unpaid:=ds.Invoices.query("status=:1"; "未払い")  
  
// これらのエンティティセレクションの参照をワーカーに引数として渡します  
CALL WORKER("mailing"; "sendMails"; $paid; $unpaid)
```

`sendMails` メソッドのコードです:

```

#DECLARE ($paid : cs.InvoicesSelection; $unpaid : cs.InvoicesSelection)
var $invoice : cs.InvoicesEntity

var $server; $transporter; $email; $status : Object

// メールの準備
$server:=New object()
$server.host:="exchange.company.com"
$server.user:="myName@company.com"
$server.password:="my!@password"
$transporter:=SMTP New transporter($server)
$email:=New object()
$email.from:="myName@company.com"

// エンティティセレクションをループします
For each($invoice;$paid)
    $email.to:=$invoice.customer.address // 顧客のメールアドレス
    $email.subject:="請求書 # "+String($invoice.number) + "のお支払いを確認いたしました。"
    $status:=$transporter.send($email)
End for each

For each($invoice;$unpaid)
    $email.to:=$invoice.customer.address // 顧客のメールアドレス
    $email.subject:="請求書 # "+String($invoice.number) + "のお支払いが確認できません。"
    $status:=$transporter.send($email)
End for each

```

エンティティセレクションとストレージ属性

すべてのストレージ属性 (テキスト、数値、ブール、日付) はエンティティセレクションの、あるいはエンティティのプロパティとして利用可能です。エンティティセレクションと組み合わせて使用した場合、スカラー属性はスカラー値のコレクションを返します。たとえば:

```

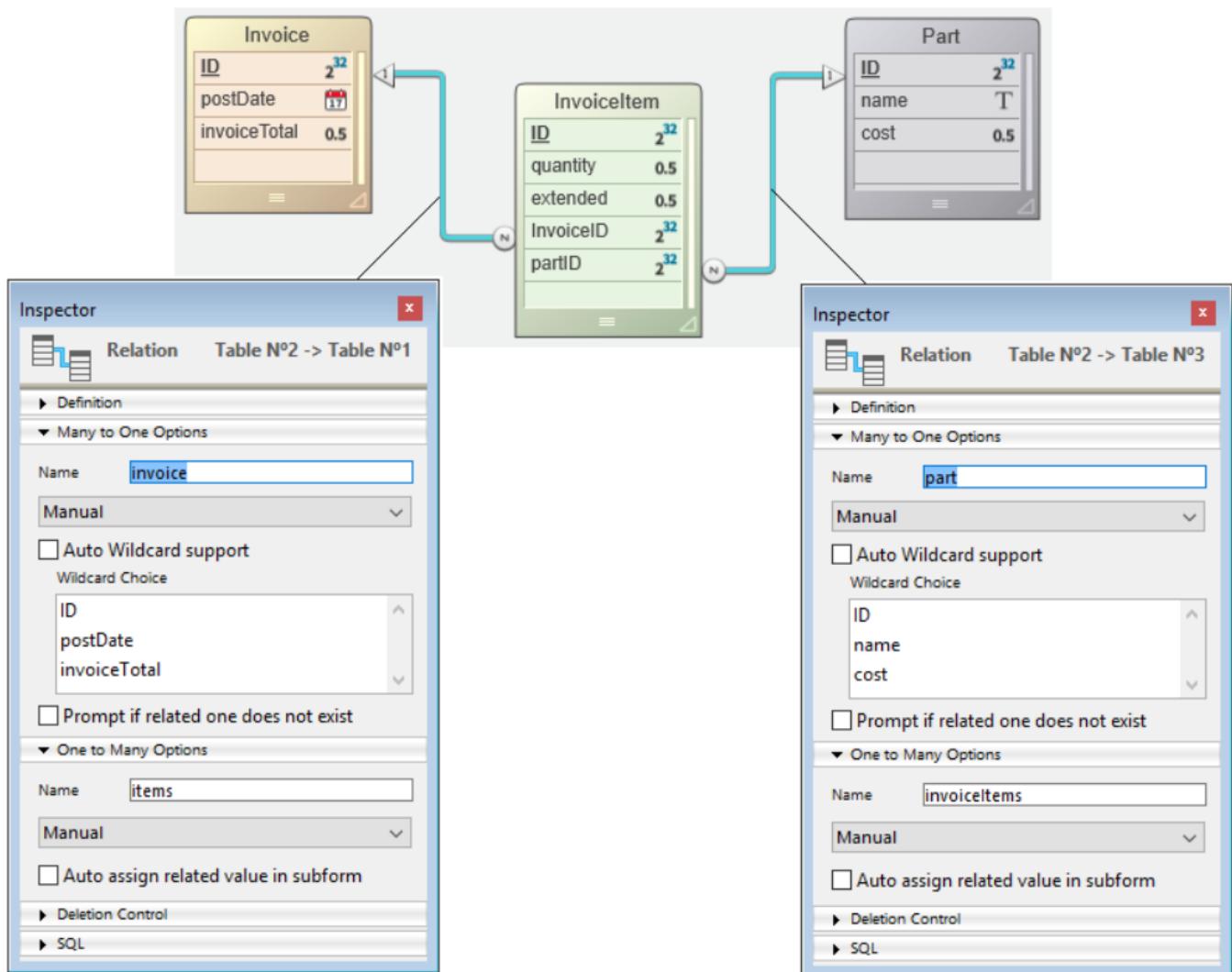
$locals:=ds.Person.query("city = :1;"+"San Jose") // 個人のエンティティセレクション
$localEmails:=$locals.emailAddress // メールアドレス（文字列）のコレクション

```

このコードは `$localEmails` 内に文字列としてのメールアドレスのコレクションを返します。

エンティティセレクションとリレーション属性

様々なクエリの方法に加えて、リレーション属性をエンティティセレクションのプロパティとして使用することで新しいエンティティセレクションを得ることもできます。たとえば、以下のようなストラクチャーの場合を考えます:



```
$myParts:=ds.Part.query("ID < 100") // ID が 100未満のpartsを返します
$myInvoices:=$myParts.invoiceItems.invoice
// $myParts 内のpartsにリレートされている請求項目を1行以上含んでいるすべての請求書
```

最後の行は、\$myParts エンティティセレクション内のpartsにリレートされている請求項目が少なくとも1行含まれているすべての請求書のエンティティセレクションを、\$myInvoices 内に返します。エンティティセレクションのプロパティとしてリレーション属性が使用されると、返される結果は、たとえ返されるエンティティが一つだけだとしても、常に新しいエンティティセレクションとなります。エンティティセレクションのプロパティとしてリレーション属性が使用された結果、エンティティが何も返ってこない場合には、返されるのは空のエンティティセレクションであり、null ではありません。

エンティティロック

一般的に、複数のユーザーあるいはプロセスが同じエンティティを同時に読み込んで変更しようとした際にコンフリクトが発生する可能性を管理する必要があります。レコードロックは、リレーションナルデータベースにおいてデータに矛盾した更新がなされないようにするための手段です。読み込み時にレコードをロックして他のプロセスが更新できないようにする、あるいは逆に保存時に読み込んでからの間に他のプロセスがレコードを変更していないかどうかを検証する、というのが基本的な概念です。前者は ベシミスティック・レコードロック と呼ばれ、他のユーザーに対してレコードをロックすることで、変更したいレコードを書き込むことができるようになります。後者は オプティミスティック・レコードロック と呼ばれ、レコードが更新される必要がある場合にのみ書き込み権限を与えるという柔軟性があります。ベシミスティック・レコードロックでは、更新される必要がないときでもレコードはロックされたままです。オプティミスティック・レコードロックではレコードの書き込みの可能/不可能は更新時に判断されます。

ORDA では、以下の二つのロックモードを提供しています：

- 自動的な "オプティミスティック" モード。多くのアプリケーションに適しています。
- "ベシミスティック" モード。エンティティをアクセスする前にロックすることができます。

自動オプティミスティック・ロック

この自動機構は、"オプティミスティック・ロック" に基づいたもので、これは Web アプリケーションの場合に特に適しています。この概念は以下のような動作

原理に基づいています:

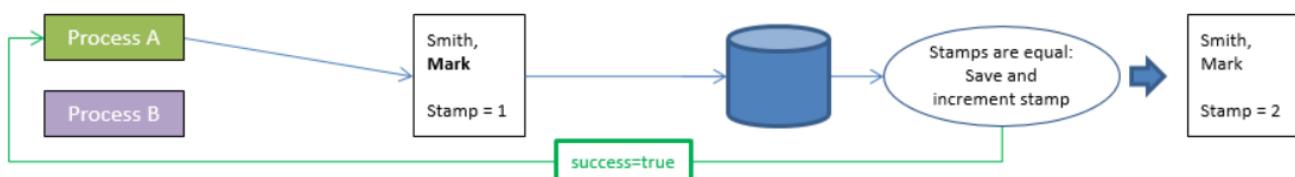
- すべてのエンティティは必ず読み書き可能な状態でロードされます。エンティティの 事前 ロックというのはありません。
- 各エンティティには保存されるたびにインクリメントされる内部的なロックスタンプを持っています。
- プロセスあるいはユーザーが `entity.save()` メソッドでエンティティを保存しようとした場合、4D は保存しようとしているエンティティのスタンプの値とデータ内にあるエンティティのスタンプの値を比較します (データ編集の場合):
 - 値が合致している場合、エンティティは保存され、内部スタンプの値はインクリメントされます。
 - 値が合致しない場合、読み込みから保存までの間に他のユーザーがエンティティを編集したことになります。保存は実行されず、エラーが返されます。

オプティミスティック・ロックの動作は以下のように図解することができます:

- 二つのプロセスが同じエンティティを読み込んだとします。



- 最初のプロセスがエンティティを編集し、それを保存しようとします。すると `entity.save()` メソッドが呼び出されます。4Dエンジンは、編集されたエンティティの内部スタンプ値とデータに保存されているエンティティの内部スタンプ値を自動的に比較します。これは合致しますので、エンティティは保存され、その内部スタンプ値はインクリメントされます。



- 二つ目のプロセスも読み込んだエンティティを編集し、それを保存しようとします。すると `entity.save()` メソッドが呼び出されます。編集されたエンティティの内部スタンプ値はデータに保存されているエンティティの内部スタンプ値と合致しないので、保存は実行されず、エラーが返されます。



この流れは以下のコードのように分解することもできます:

```
$person1:=ds.Person.get(1) // エンティティを参照
$person2:=ds.Person.get(1) // 同じエンティティへの別の参照
$person1.name:="Bill"
$result:=$person1.save() // $result.success=true, 変更は保存されます
$person2.name:="William"
$result:=$person2.save() // $result.success=false, 変更は保存されません
```

この例では、\$person1 に Person の、キーが 1 のエンティティを代入します。次に、同じエンティティの別の参照を変数 \$person2 に代入します。\$person1 を用いて、人物の名前を変更してエンティティを保存します。同じことを \$person2 を使用して実行しようとすると、4D はディスク上のエンティティをチェックし、変数 \$person2 に代入されたものと同じかどうかを調べます。結果としてこれは同じものでは無いので、success プロパティには false が返され、二つ目の変更は保存されません。

こういった状況が発生した場合には、たとえば `entity.reload()` メソッドを使用してディスクからエンティティを再読込し、変更をもう一度おこなうことができます。また `entity.save()` メソッドは、異なるプロセスがそれぞれ異なる属性を変更していた場合には保存を実行する "automerge" オプションも提供しています。

トランザクション 内においては、特定レコードのコピーがコンタキスト内に一つしか存在しないため、レコードスタンプは使用されません。レコードを参照するエンティティが複数あっても、同じコピーが検収されるため `entity.save()` による処理がスタンプエラーを生成することはありません。

ペシミスティック・ロック

エンティティは、データアクセス時に任意にロックおよびアンロックすることができます。エンティティがプロセスからロックされている場合、そのエンティティはプロセスに読み書き可能モードで読み込まれていますが、他のすべてのプロセスに対してロックされています。ロックされたエンティティは、他のプロセスからは読み込みモードでのみ読み込むことができます。つまり、その値を編集・保存することはできません。

この機能は `Entity` クラスの 2つの関数に基づいています:

- `entity.lock()`
- `entity.unlock()`

詳細な情報については、これらの関数の説明を参照してください。

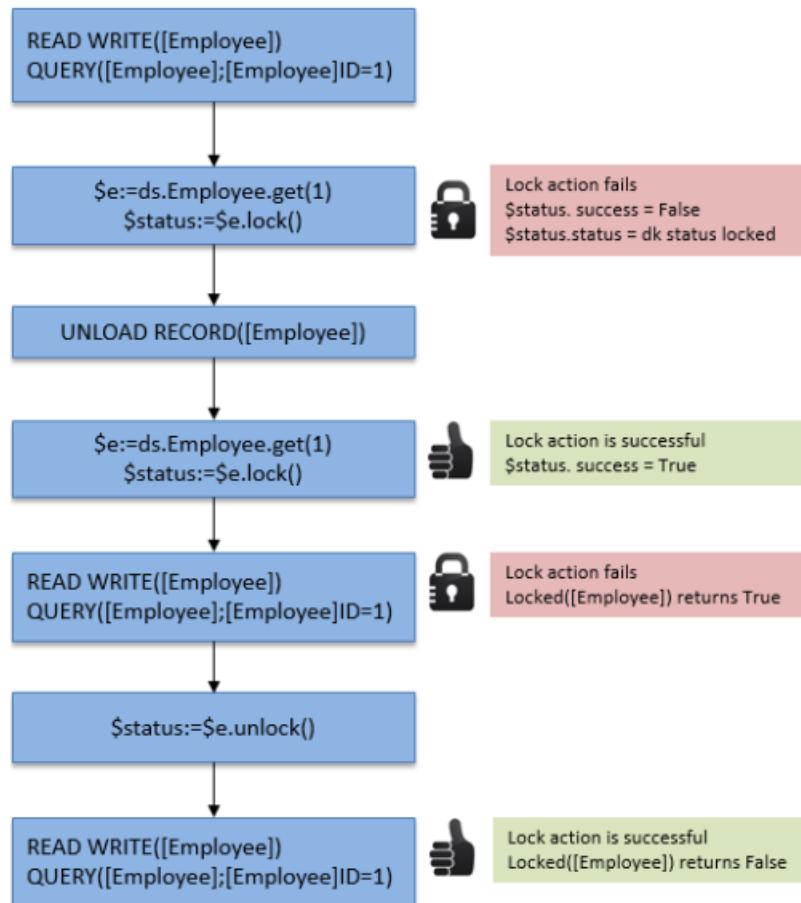
ペシミスティック・ロックは [REST API](#) によって扱うことも可能です。

4Dクラシック・ロックとORDAのペシミスティック・ロックの組み合わせ

クラシックコマンドと ORDA コマンドの両方を使用してレコードをロックする場合、以下の原則に注意する必要があります:

- クラシック4Dコマンドを使用してレコードをロックした場合、そのレコードに相当するエンティティを ORDA でロックすることはできません。
- ORDA を使用してエンティティをロックした場合、そのエンティティに相当するレコードをクラシック4Dコマンドでロックすることはできません。

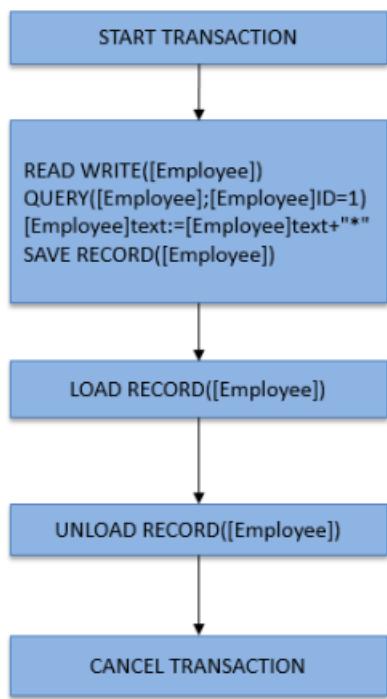
これらの原理は以下のような図に表すことができます:



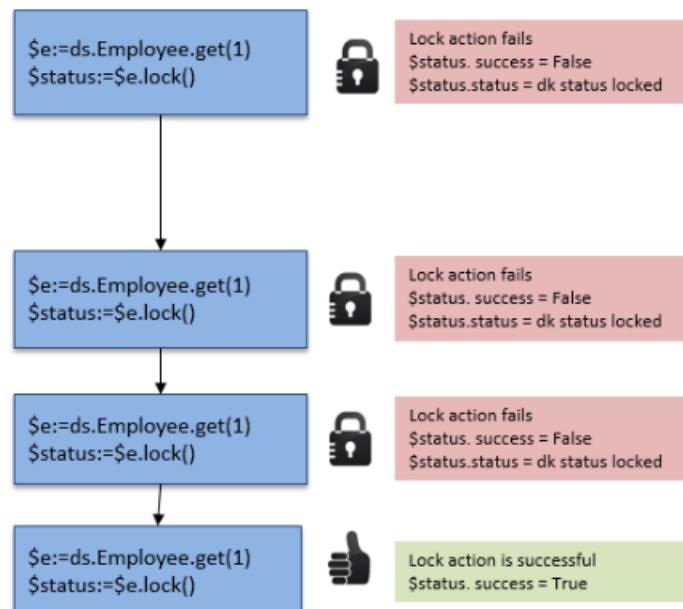
トランザクションロックはクラシックコマンドと ORDAコマンドの両方に適用されます。マルチプロセスあるいはマルチユーザーアプリケーションにおいては、トランザクション内でクラシックコマンドを使用してレコードをロックした場合、そのトランザクションが OK あるいはキャンセルされるまで、他のプロセスからそのコードに相当するエンティティをロックすることはできません（逆もまた然りです）。

- クラシックコマンドを使用してロックした場合:

Process P1

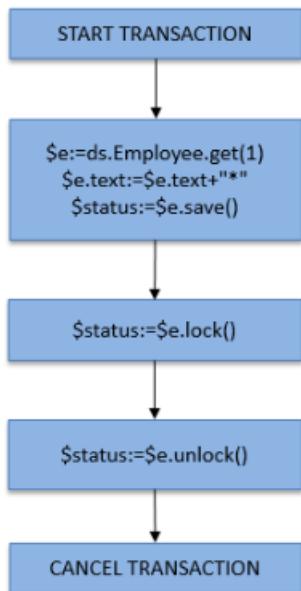


Process P2

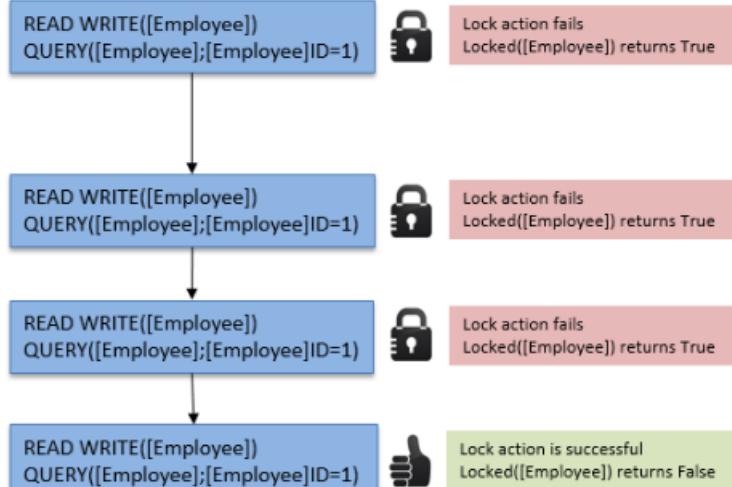


- ORDA関数を使用してロックした場合:

Process P1



Process P2



リモートデータストアの利用

4D アプリケーション上で公開された [データストア](#) は、異なるクライアントにより同時にアクセスすることができます：

- 4D リモートアプリケーションは ORDA を使っていれば、`ds` コマンドでメインデータストアにアクセスできます。この 4D リモートアプリケーションは従来のモードでもデータベースにアクセスできます。これらのアクセスを処理するのは 4D アプリケーションサーバー です。
- 他の 4D アプリケーション (4Dリモート、4D Server) は、[Open datastore](#) コマンドを使ってリモートデータストアのセッションを開始できます。アクセスを処理するのは HTTP REST サーバー です。
- モバイルアプリケーションを更新するための [4D for iOS](#) または [4D for Android](#) のクエリでアクセスできます。アクセスを処理するのは HTTP サーバー です。

セッションの開始

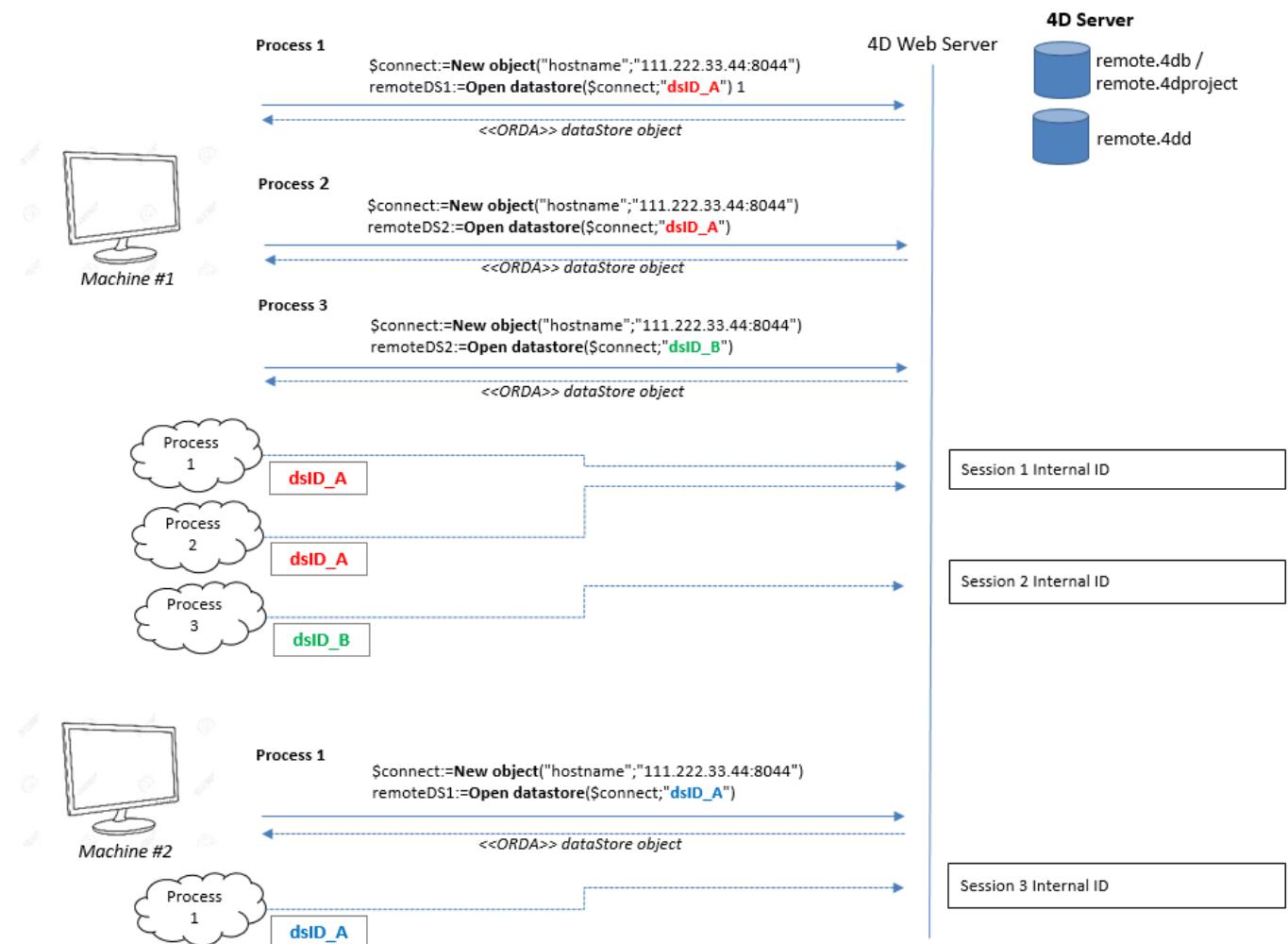
`Open datastore` コマンドによって参照されるリモートデータストアの場合、リクエスト元プロセスとリモートデータストア間の接続はセッションにより管理されます。

リモートデータストア上では、接続を管理するためのセッションが作成されます。このセッションは内部的にセッションID によって識別され、4D アプリケーション上では `localID` と紐づいています。データ、エンティティセレクション、エンティティへのアクセスはこのセッションによって自動的に管理されます。

`localID` はリモートデータストアに接続しているマシンにおけるローカルな識別IDです：

- 同じアプリケーションの別プロセスが同じリモートデータストアに接続する場合、`localID` とセッションは共有することができます。
- 同じアプリケーションの別プロセスが別の `localID` を使って同じデータストアに接続した場合、リモートデータストアでは新しいセッションが開始されます。
- 他のマシンが同じ `localID` を使って同じデータストアに接続した場合、新しいセッションが新しい cookie で開始されます。

これらの原則を下図に示します：



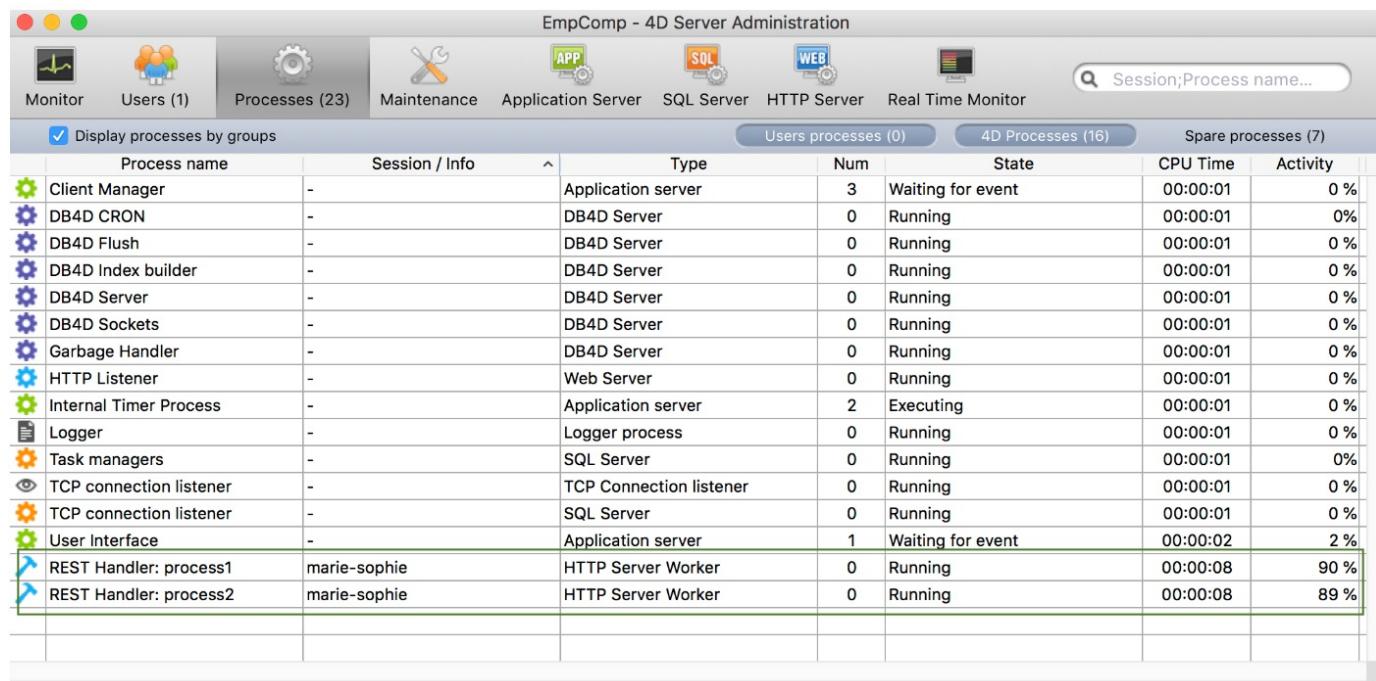
REST リクエストによって開かれたセッションについては、[ユーザーとセッション](#) を参照ください。

セッションの監視

データストアアクセスを管理しているセッションは 4D Server の管理ウィンドウに表示されます:

- プロセス名: "REST Handler: <process name>"
- タイプ: HTTP Server Worker
- セッション: Open datastore コマンドに渡されたユーザー名

次の例では、1つのセッション上で 2つのプロセスが実行中です:



EmpComp - 4D Server Administration									
Monitor		Users (1)	Processes (23)	Maintenance	Application Server	SQL Server	HTTP Server	Real Time Monitor	Session;Process name...
Display processes by groups		Users processes (0)		4D Processes (16)		Spare processes (7)			
Process name	Session / Info	Type	Num	State	CPU Time	Activity			
Client Manager	-	Application server	3	Waiting for event	00:00:01	0 %			
DB4D CRON	-	DB4D Server	0	Running	00:00:01	0 %			
DB4D Flush	-	DB4D Server	0	Running	00:00:01	0 %			
DB4D Index builder	-	DB4D Server	0	Running	00:00:01	0 %			
DB4D Server	-	DB4D Server	0	Running	00:00:01	0 %			
DB4D Sockets	-	DB4D Server	0	Running	00:00:01	0 %			
Garbage Handler	-	DB4D Server	0	Running	00:00:01	0 %			
HTTP Listener	-	Web Server	0	Running	00:00:01	0 %			
Internal Timer Process	-	Application server	2	Executing	00:00:01	0 %			
Logger	-	Logger process	0	Running	00:00:01	0 %			
Task managers	-	SQL Server	0	Running	00:00:01	0 %			
TCP connection listener	-	TCP Connection listener	0	Running	00:00:01	0 %			
TCP connection listener	-	SQL Server	0	Running	00:00:01	0 %			
User Interface	-	Application server	1	Waiting for event	00:00:02	2 %			
REST Handler: process1	marie-sophie	HTTP Server Worker	0	Running	00:00:08	90 %			
REST Handler: process2	marie-sophie	HTTP Server Worker	0	Running	00:00:08	89 %			

ロックとトランザクション

エンティティロックやトランザクションに関連した ORDA 機能は、ORDA のクライアント / サーバーモードと同様に、リモートデータストアにおいてもプロセスレベルで管理されます:

- あるプロセスがリモートデータストアのエンティティをロックした場合、セッションの共有如何に関わらず、他のすべてのプロセスに対してそのエンティティはロックされた状態です ([エンティティロック](#) 参照)。同一のレコードに対応する複数のエンティティが 1つのプロセスによってロックされている場合、同プロセス内でそれらがすべてアンロックされないと、ロックは解除されません。なお、ロックされたエンティティに対する参照がメモリ上に存在しなくなった場合にも、ロックは解除されます。
- トランザクションは `dataStore.startTransaction()`、`dataStore.cancelTransaction()`、`dataStore.validateTransaction()` のメソッドを使って、リモートデータストアごとに個別に開始・認証・キャンセルすることができます。これらの操作は他のデータストアには影響しません。
- 従来の 4D ランゲージコマンド (`START TRANSACTION` , `VALIDATE TRANSACTION` , `CANCEL TRANSACTION`) は `ds` で返されるメインデータストアに対してのみ動作します。リモートデータストアのエンティティがあるプロセスのトランザクションで使われている場合、セッションの共有如何に関わらず、他のすべてのプロセスはそのエンティティを更新できません。
- 次の場合にエンティティのロックは解除され、トランザクションはキャンセルされます:
 - プロセスが強制終了された
 - サーバー上でセッションが閉じられた
 - サーバー管理ウィンドウからセッションが強制終了された

セッションの終了

アクティビティなしにタイムアウト時間が経過すると、4D は自動的にセッションを終了します。デフォルトのタイムアウト時間は 60 分です。`Open datastore` コマンドの `connectionInfo` パラメーターを指定して、タイムアウト時間を変更することができます。

セッション終了後にリクエストがリモートデータストアに送信された場合、セッションは可能な限り (ライセンスがあり、サーバーが停止していないなど) 再開

されます。ただしセッションが再開しても、ロックやトランザクションに関わるコンテキストは失われていることに留意が必要です（前述参照）。

クライアント/サーバーの最適化

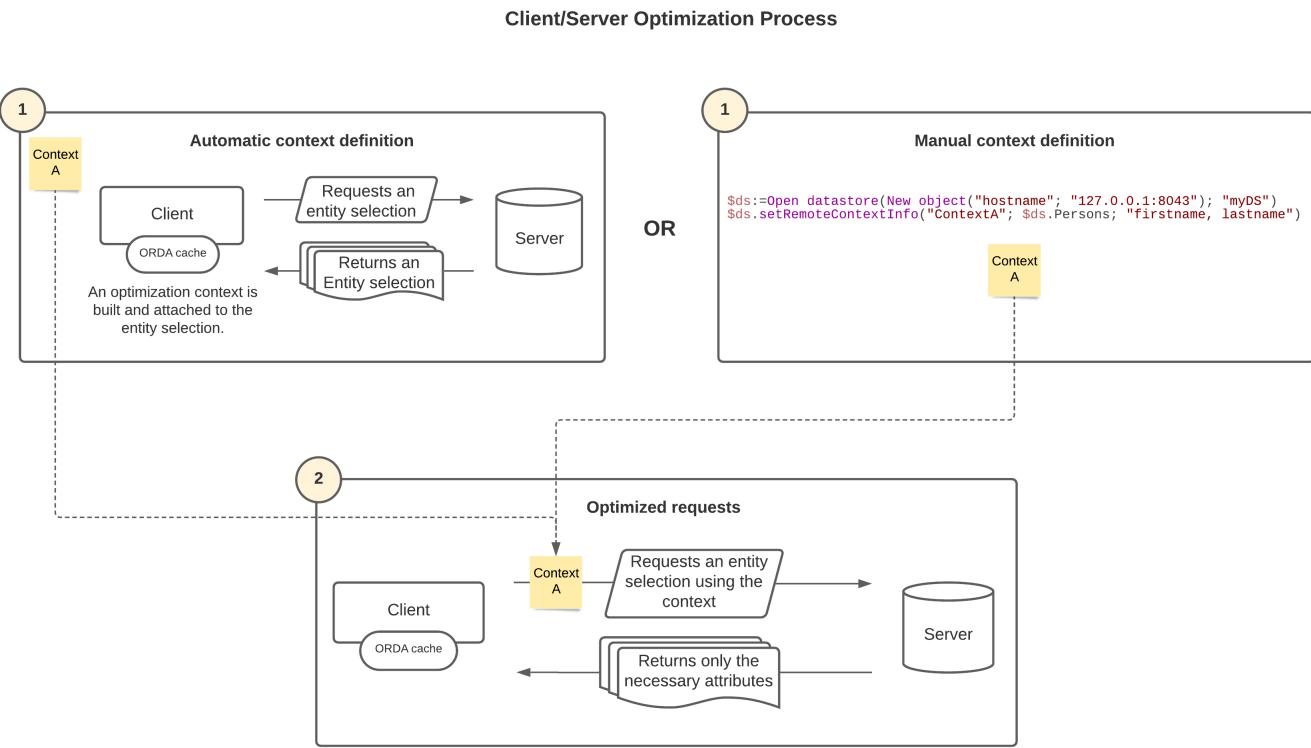
4Dは、クライアント/サーバー環境において（`ds` または `Open datastore` によりアクセスされたデータストアの場合）エンティティセレクションを使用、あるいはエンティティを読み込む ORDAリクエストについて最適化する機構を提供しています。この最適化機構は、ネットワーク間でやり取りされるデータの量を大幅に縮小させることで 4Dの実行速度を向上させます。これには以下のような機能が含まれます：

- 最適化コンテキスト
- ORDAキャッシュ

コンテキスト

最適化コンテキストは、以下の実装に基づいています：

- クライアントがサーバーに対してエンティティセレクションのリクエストを送ると、4D はコード実行の途中で、エンティティセレクションのどの属性がクライアント側で実際に使用されているかを自動的に "学習" し、それに対応した "最適化コンテキスト" をビルドします。このコンテキストはエンティティセレクションに付随し、使用された属性を保存していきます。他の属性があとで使用された場合には自動的に情報を更新していきます。以下のメソッドや関数が学習のトリガーとなります：
 - `Create entity selection`
 - `dataClass.fromCollection()`
 - `dataClass.all()`
 - `dataClass.get()`
 - `dataClass.query()`
 - `entitySelection.query()`
- サーバー上の同じエンティティセレクションに対してその後に送られたリクエストは、最適化コンテキストを再利用して、サーバーから必要な属性のみを取得していくことで、処理を速くします。たとえば、[エンティティセレクション型のリストボックス](#) では、先頭行の表示中に学習がおこなわれます。次の行からは、表示が最適化されます。以下の関数は、ソースのエンティティセレクションの最適化コンテキストを、戻り値のエンティティセレクションに自動的に付与します：
 - `entitySelection.and()`
 - `entitySelection.minus()`
 - `entitySelection.or()`
 - `entitySelection.orderBy()`
 - `entitySelection.slice()`
 - `entitySelection.drop()`
- 既存の最適化コンテキストは、同じデータクラスの他のエンティティセレクションであればプロパティとして渡すことができるので、学習フェーズを省略して、アプリケーションをより速く実行することができます（以下の [contextプロパティの使用](#) を参照してください）。
- `dataStore.setRemoteContextInfo()` 関数を使用して、最適化コンテキストを手動で構築することができます（[コンテキストの事前設定](#) 参照）。



例題

以下のようなコードがあるとき:

```
$sel:=$ds.Employee.query("firstname = ab@")
For each($e;$sel)
    $s:=$e.firstname+" "+$e.lastname+" works for "+$e.employer.name // $e.employer は Company テーブルを参照
End for each
```

最適化機構のおかげでこのリクエストは、ループの 2回目の繰り返しより、\$sel の中に実際に使用されている属性 (firstname, lastname, employer, employer.name) のデータのみを取得するようになります。

contextプロパティの再利用

context プロパティを使用することで、最適化の利点をさらに増幅させることができます。このプロパティは、あるエンティティセレクション用に "学習した" 最適化コンテキストを参照します。これを新しいエンティティセレクションを返す ORDA関数に引数として渡すことで、その返されたエンティティセレクションでは学習フェーズを最初から省略して使用される属性をサーバーにリクエストできるようになります。

`.setRemoteContextInfo()` 関数を使って、コンテキストを作成することもできます。

エンティティセレクションを扱うすべての ORDA関数は、context プロパティをサポートします (たとえば `dataClass.query()` あるいは `dataClass.all()` など)。同じ最適化 context プロパティは、同じデータクラスのエンティティセレクションに対してであればどのエンティティセレクションにも渡すことができます。ただし、コードの他の部分で新しい属性が使用された際にはコンテキストは自動的に更新されるという点に注意してください。同じコンテキストを異なるコードで再利用しすぎると、コンテキストを読み込み過ぎて、結果として効率が落ちる可能性があります。

同様の機構は読み込まれたエンティティにも実装されており、それによって使用した属性のみがリクエストされるようになります (`dataClass.get()` 関数参照)。

`dataClass.query()` を使用した例:

```

var $sel1; $sel2; $sel3; $sel4; $querysettings; $querysettings2 : Object
var $data : Collection
$querysettings:=New object("context";"shortList")
$querysettings2:=New object("context";"longList")

$sel1:=ds.Employee.query("lastname = S@";$querysettings)
$data:=extractData($sel1) // extractData メソッド内で最適化がトリガーされ、
// コンテキスト "shortList" に紐づけられます

$sel2:=ds.Employee.query("lastname = Sm@";$querysettings)
$data:=extractData($sel2) // extractData メソッド内で最適化がトリガーされ、
// コンテキスト "shortList" に紐づけられます

$sel3:=ds.Employee.query("lastname = Smith";$querysettings2)
$data:=extractDetailedData($sel3) // extractDetailedData メソッド内で最適化がトリガーされ、
// コンテキスト "longList" に紐づけられます

$sel4:=ds.Employee.query("lastname = Brown";$querysettings2)
$data:=extractDetailedData($sel4) // extractDetailedData メソッド内で最適化がトリガーされ、
// コンテキスト "longList" に紐づけられます

```

エンティティセレクション型リストボックス

クライアント/サーバー環境におけるエンティティセレクション型リストボックスにおいては、そのコンテンツを表示またはスクロールする際に、最適化が自動的に適用されます。つまり、リストボックスに表示されている属性のみがサーバーにリクエストされます。

また、リストボックスの カレントの項目 プロパティ式 ([コレクション/エンティティセレクション型リストボックス 参照](#)) を介してカレントエンティティをロードする場合には、専用の "ページモード" コンテキストが提供されます。これによって、"ページ" が追加属性をリクエストしても、リストボックスのコンテキストのオーバーロードが避けられます。なお、ページコンテキストの生成/使用は カレントの項目 式を使用した場合に限ります (たとえば、`entitySelection[index]` を介して同じエンティティにアクセスした場合は、エンティティセレクションコンテキストが変化します)。

その後、エンティティを走査する関数がサーバーに送信するリクエストにも、同じ最適化が適用されます。以下の関数は、ソースエンティティの最適化コンテキストを、戻り値のエンティティに自動的に付与します:

- `entity.next()`
- `entity.first()`
- `entity.last()`
- `entity.previous()`

たとえば、次のコードは選択したエンティティをロードし、所属しているエンティティセレクションを走査します。エンティティは独自のコンテキストにロードされ、リストボックスのコンテキストは影響されません:

```

$myEntity:=Form.currentElement // カレントの項目式
//... なんらかの処理
$myEntity:=$myEntity.next() // 次のエンティティも同じコンテキストを使用してロードされます

```

コンテキストの事前設定

最高のパフォーマンスを得るには、アプリケーションの機能またはアルゴリズムごとに最適化コンテキストが定義されていることが推奨されます。たとえば、あるコンテキストは顧客に関するクエリに、別のコンテキストは商品に関するクエリに使用することができます。

最高レベルに最適化された最終アプリケーションを提供したい場合、以下の手順でコンテキストを事前に設定することで、学習フェーズを省略することができます:

1. アルゴリズムを設計します。
2. アプリケーションを実行し、自動学習メカニズムに最適化コンテキストを作成させます。
3. `dataStore.getRemoteContextInfo()` または `dataStore.getAllRemoteContexts()` 関数を呼び出して、コンテキストを収集します。`entitySelection.getRemoteContextAttributes()` と `entity.getRemoteContextAttributes()` 関数を使用して、アルゴリズムがどのように属性を使用するかを分析することができます。
4. 最後に、アプリケーション起動時に `dataStore.setRemoteContextInfo()` 関数を呼び出してコンテキストを構築し、これらをアルゴリズムで

使用します。

ORDAキャッシング

最適化のため、ORDA経由でサーバーにリクエストしたデータは、(4Dキャッシングとは異なる) ORDAリモートキャッシングに読み込まれます。ORDAキャッシングはデータクラスごとに構成され、30秒後に失効します。

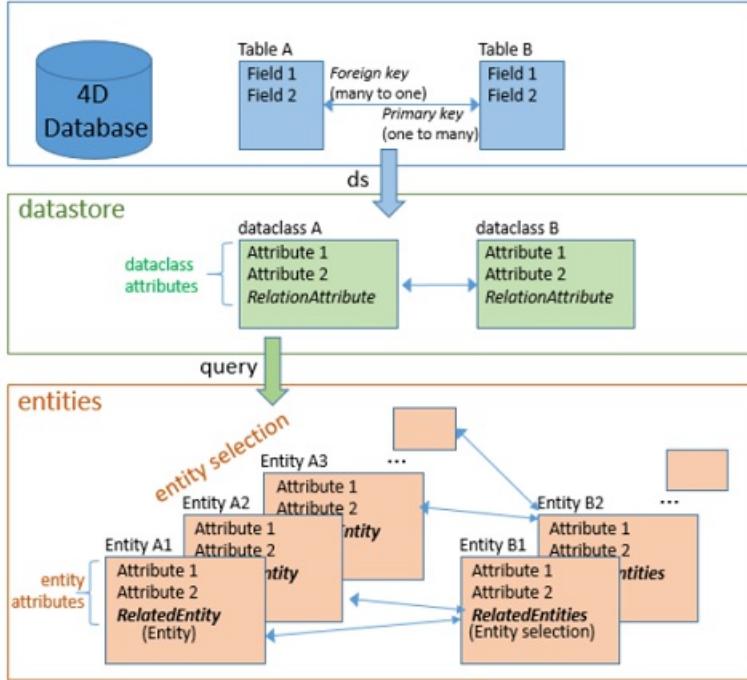
キャッシングに含まれるデータは、タイムアウトに達すると期限切れとみなされます。期限切れデータにアクセスする場合は、サーバーにリクエストが送信されます。期限切れデータは、スペースが必要になるまでキャッシングに残ります。

デフォルトでは、ORDAキャッシングは 4D によって透過的に処理されます。しかし、以下の ORDAクラスの関数を使用して、その内容を制御することができます:

- `dataClass.setRemoteCacheSettings()`
- `dataClass.getRemoteCache()`
- `dataClass.clearRemoteCache()`

用語集

主なコンセプトの概要



属性

属性とは、リレーションナルデータベース内における最小の保存セルです（[リレーション属性](#)も参照してください）。データクラス属性とエンティティ属性を混同しないようにしてください：

- データクラスオブジェクトにおける各プロパティは、対応するテーブルの対応するフィールドへとマップするデータクラス属性です（同じ名前と型）。
- エンティティオブジェクトにおけるエンティティ属性は、対応するデータストア属性の値を格納するプロパティです。

属性とプロパティは似た概念です。"属性"はデータを保存するデータクラスプロパティを指定するのに使われるのに対し、"プロパティ"はより一般的な概念でオブジェクト内で保存されるデータを定義します。

属性パス

属性パスとは、あるデータクラスあるいはエンティティ内の属性へのパスです。[プロパティパス](#)も参照してください。

クラスコード

ユーザークラス関数のコード。

計算属性

計算属性は、実際には情報を保存しません。代わりに、同じエンティティや他のエンティティ、属性、関数などから得られる値に基づいて、その属性値を決定します。計算属性が参照されると、それを定義する"計算"が評価され、値が決定されます。計算属性の値を算出するにあたっては、ユーザー定義のコードによって、その評価中に評価の仕方を決定することも可能です。

データモデルクラス

データモデルオブジェクトに関連して提供される拡張クラス。

データモデルオブジェクト

ORDA を通じて提供されているデータベースオブジェクト (データストア、データクラス、エンティティ、エンティティセレクション)。

データモデル関数

ORDA データモデルクラスの関数。

データクラス

データクラスとは、データを記述するオブジェクトモデルです。データストアによって提供されるデータベースのテーブルは、データクラスを通して管理されます。データストアから提供されたデータベースの各テーブルは、対応する同名のデータクラスを持ちます。テーブルの各フィールドは、データクラスの属性です。

データクラスは単一のデータストアにリレートされています。

DataClass クラス

カスタム関数を追加することのできる、特定のデータクラスオブジェクト用のクラス。

データストア

データストアとは、ORDA によって提供されるインターフェースオブジェクトです。データストアはストラクチャーを参照し、データへのアクセスを提供します。

`ds` コマンドによって返されるメインデータベースは、データストア (メインデータストア) として利用可能です。

データストアは以下のものを提供します:

- 4Dデータベースへの接続
- データベースを扱うためのデータクラスのセット

利用できるデータベースは、スタンダロンまたはクライアント/サーバーで開いているメインの 4D データベース (メインデータストア)、および REST リソースとして公開された 4D Server データベースです (リモートデータストア)。

データストアは単一のデータベースのみを参照しますが、複数のデータベースをアクセスするために複数のデータストアを開くことも可能です。

DataStore クラス

カスタム関数を追加することのできる、データストアオブジェクト用のクラス。

DataStoreImplementation

`4D` クラスストア内の DataStore クラスの内部的な名称。

ディープコピー

ディープコピーは、あるオブジェクトとそこに格納されているすべての参照を複製します。ディープコピーのあと、コピーされたコレクションには、すべてのオリジナル要素の複製 (つまり新規参照) が格納されています。[シャロウコピー](#) も合わせて参照してください。

ds

`ds` は、[データストア](#) のオブジェクト参照を返す 4D ランゲージコマンドです。この参照は 4D のメインデータベースが提供するデータストアに合致します。

エンティティ

エンティティとは、データクラスモデルに対応するオブジェクトです。エンティティには、データクラスと同じ属性が格納されます。

エンティティは、データクラスのインスタンスとも解釈可能なオブジェクトです。しかしながら、エンティティはリレートされたデータも格納しています。エンティティの目的はデータの管理（作成、更新、削除）です。

詳細な情報については、[エンティティ](#) を参照してください。

エンティティセレクション

エンティティセレクションは、一つのオブジェクトです。データストアをクエリすると、エンティティセレクションが返されます。エンティティセレクションとは、同じデータクラスに所属するエンティティへの参照のセットのことです。

エンティティセレクションは以下を格納します：

- 0 から X までのエンティティ参照のセット
- length プロパティ（常に存在します）
- queryPlan および queryPath プロパティ（クエリ時に要求した場合に存在します）

エンティティセレクションは空であることもあります。

汎用クラス

エンティティやデータクラスなどの ORDA オブジェクト用のビルトインクラス。汎用クラスのプロパティや関数は、ユーザー拡張クラス（例：`EmployeeEntity`）において自動で利用可能です。

レイジーローディング

エンティティは参照として管理されているため、データは必要なときにのみロードされます。つまりコードや、インターフェースウィジェットを通してアクセスしたときなどです。この最適化原理は、レイジーローディングと呼ばれています。

メインデータストア

開かれている 4D データベース（シングルユーザーまたはクライアント/サーバー）に対応するデータストアオブジェクト。メインデータストアは `ds` コマンドによって返されます。

メソッド

データストア、データクラス、エンティティセレクション、エンティティなどの ORDA オブジェクトは、オブジェクトのクラスを定義します。これらのクラスには、オブジェクトを直接操作するための専用のメソッドが提供されています。これらのメソッドはメンバー関数とも呼ばれます。このメソッドを使用するには、オブジェクトのインスタンスに対して呼び出します。

たとえば、`query()` メソッドはデータクラスのメンバー関数です。`$myClass` 変数にデータクラスオブジェクトを格納している場合、次のように書くことができます：

```
$myClass.query("name = smith")
```

ミックスデータ型

このドキュメントでは、データクラス属性に保存可能な値の様々な型を指定するために、"ミックス" データ型が使用されます。：

- number
- テキスト
- null
- boolean

- date
- object
- collection
- ピクチャー (*)

(*) ピクチャー型は `entitySelection.max()` などの統計型メソッドではサポートされていません。

オプティミスティック・ロック

"オプティミスティック・ロック" モードでは、エンティティは更新されるまでは明示的にはロックされていません。各エンティティは、そのエンティティがディスクに保存されるたびに自動でインクリメントされる内部スタンプを持っています。`entity.save()` および `entity.drop()` メソッドは(メモリ内に) ロードされたスタンプと、ディスク上のエンティティのスタンプが合致しない場合、あるいはエンティティがドロップされている場合にはエラーを返します。オプティミスティック・ロックは ORDA 実装内でのみ使用可能です。[ペシミスティック・ロック](#) も合わせて参照してください。

ペシミスティック・ロック

"ペシミスティック・ロック" とは、`entity.lock()` メソッドにより、エンティティがアクセスされる前にそれをロックすることを意味します。ロックが解除されるまで、他のプロセスからはそのエンティティを更新することも、ドロップすることもできません。クラシック 4D ランゲージにおいてはペシミスティック・ロックのみが利用可能です。[オプティミスティック・ロック](#) も合わせて参照してください。

プロパティ

[属性](#) を参照してください。

属性とプロパティは似た概念です。"属性" はデータを保存するデータクラスプロパティを指定するのに使われるのに対し、"プロパティ" はより一般的な概念でオブジェクト内で保存されるデータを定義します。

プロパティパス

プロパティパスとは、あるオブジェクトのプロパティへのパスです。プロパティが複数の階層にネストされている場合、各階層はドット (".") によって区切られます。

通常クラス

ORDA オブジェクトとは関わりのないユーザークラス。

リレートされたデータクラス

リレートされたデータクラスとは、リレーション属性によってリンクされたデータクラスのことを指します。

リレーション属性

リレーション属性は、データクラス間のリレーション(1対N および N対1)を概念化するものです。

- N対1リレーション(データクラスA はデータクラスB のオカレンスを参照します): リレーション属性はデータクラスA 内で利用可能で、データクラスB の一つのインスタンスを参照します。
- 1対Nリレーション(データクラスB のオカレンスがデータクラスA の複数のオカレンスを参照します): リレーション属性はデータクラスB 内で利用可能で、データクラスA の複数のインスタンスを参照します。

データクラスは再帰的なリレーション属性を持つことができます。

エンティティ内では、リレーション属性の値はエンティティあるいはエンティティセレクションとなります。

リレートエンティティ

リレートエンティティはデータクラス内のリレーション属性のインスタンスとしてみることができます。

エンティティセレクションは、対応するデータクラス内で定義されたリレーション属性に応じて、複数のリレートエンティティを参照することもあります。

リモートデータストア

4D または (HTTP経由で利用可能な) 4D Server 上で開かれている、REST リソースとして公開された 4Dデータベース。このデータベースは他のマシンにおいてデータストアとしてローカルに参照することができ、その際には割り当てられた localID で識別されます。リモートデータストアは ORDA の概念 (データストア、データクラス、エンティティセレクション等) を使って利用できます。利用にあたってはライセンスが消費されます。

セッション

4Dアプリケーションがリモートデータストアに接続すると、4D Server (HTTP) 上では セッション が作成されます。セッションcookie が生成され、ローカルデータストアID と紐づけられます。

新規セッションが開始されるごとに、ライセンスが消費されます。セッションが閉じられると、ライセンスは解放されます。

アクティビティのないセッションはタイムアウト後に自動的に終了します。デフォルトのタイムアウトは 48時間で、任意に設定することができます (最少時間は 60分)。

シャロウコピー

シャロウコピーは、要素の構造のみを複製し、同じ内部参照を保持します。シャロウコピーのあと、二つのコレクションに格納された個々の要素は同じものが共有されています。[ディープコピー](#) も合わせて参照してください。

記号

"オプティミステック" ロックテクノロジーにおいて使用されるものです。すべてのエンティティにはスタンプと呼ばれる内部カウンターがあり、エンティティが保存されるたびにインクリメントされています。エンティティ内のスタンプとディスク上に保存されているエンティティのスタンプを自動的に比較することで、4D は同じエンティティへの書き込みの衝突を防いでいます。

ストレージ属性

ストレージ属性 (スカラー属性と呼ばれることも) は、データストアクラスの属性の中で最も基本的なタイプであり、リレーションナルデータベースのフィールドに最も直接的に対応するものです。ストレージ属性は、データクラスのエンティティ毎に 1つの値を持ちます。

BLOB

Blobクラスを使って、[BLOB オブジェクト](#) (`4D.Blob`) を操作することができます。

概要

`4D.Blob.new() : 4D.Blob`

`4D.Blob.new(blobScal : Blob) : 4D.Blob`

`4D.Blob.new(blobObj : 4D.Blob) : 4D.Blob`

は新規の `4D.Blob` オブジェクトを作成し、(任意) 別の BLOBデータ (スカラーBLOB または `4D.Blob`) のコピーを格納します

`.size : Real`

`4D.Blob` のサイズを返します (バイト単位)。

`.slice() : 4D.Blob`

`.slice(start : Real) : 4D.Blob`

`.slice(start : Real; end : Real) : 4D.Blob`

呼び出し対象である BLOB のデータの一部を参照する新規の `4D.Blob` を作成して返します。元の BLOB は変更されません。

4D.Blob.new()

▶ 補足

`4D.Blob.new() : 4D.Blob`

`4D.Blob.new(blobScal : Blob) : 4D.Blob`

`4D.Blob.new(blobObj : 4D.Blob) : 4D.Blob`

引数	タイプ		説明
blob	Blob or 4D.Blob	->	コピーする BLOB
戻り値	4D.Blob	<-	新規 4D.Blob

説明

`4D.Blob.new` は新規の `4D.Blob` オブジェクトを作成し、(任意) 別の BLOBデータ (スカラーBLOB または `4D.Blob`) のコピーを格納します。

`blob` 引数が渡されなかった場合、関数は空の `4D.Blob` を返します。

.size

`.size : Real`

説明

`.size` プロパティは、`4D.Blob` のサイズを返します (バイト単位)。

.slice()

▶ 補足

`.slice() : 4D.Blob`

`.slice(start : Real) : 4D.Blob`

`.slice(start : Real; end : Real) : 4D.Blob`

引数	タイプ		説明
start	Real	->	新しい 4D.Blob に含める最初のバイトのインデックス
end	Real	->	新しい 4D.Blob に含めない最初のバイトのインデックス
戻り値	4D.Blob	<-	新規 4D.Blob

説明

.slice() 関数は、呼び出し対象である BLOB のデータの一部を参照する新規の 4D.Blob を作成して返します。元の BLOB は変更されません。start 引数は、新しい 4D.Blob に含める最初のバイトを示す BLOB のインデックスです。負の値を指定した場合、4D は BLOB の末尾から先頭に向かってオフセットしたものとして扱います。たとえば、-10 は BLOB の最後から 10 番目のバイトを表します。デフォルト値は 0 です。start にソースBLOB のサイズより大きな値を指定すると、返される 4D.Blob のサイズは 0 になり、データは含まれません。

end 引数は、新しい 4D.Blob に含めない最初のバイトを示す BLOB のインデックスです。つまり、指定インデックスのバイトは新しい 4D.Blob から除外されます。負の値を指定した場合、4D は BLOB の末尾から先頭に向かってオフセットしたものとして扱います。たとえば、-10 は BLOB の最後から 10 番目のバイトを表します。デフォルト値は BLOB のサイズです。

例題

```
var $myBlob : 4D.Blob

// 4D.Blob にテキストを格納します
CONVERT FROM TEXT("Hello, World!"; "UTF-8"; $myBlob)
$is4DBlob:=OB Instance of($myBlob; 4D.Blob); // True

$myString:=Convert to text($myBlob; "UTF-8")
// $myString は "Hello, World!" を格納しています

// $myBlob から新しい 4D.Blob を作成します
$myNewBlob:=$myBlob.slice(0; 5)

$myString:=Convert to text($myNewBlob; "UTF-8")
// $myString は "Hello" を格納します
```

Class

プロジェクトにおいてユーザークラスが [定義](#) されていれば、それは 4D ランゲージ環境に読み込まれます。クラスとは、それ自身が "Class" クラスのオブジェクトであり、プロパティと関数を持ちます。

概要

.name : Text

4D.Class オブジェクトの名称を格納します

.new(param : any { ;...paramN }) : 4D.Class

対象クラスの新規インスタンスである cs.className オブジェクトを作成して返します

.superclass : 4D.Class

対象クラスの親クラスを返します

.name

▶ [履歴](#)

.name : Text

説明

.name プロパティは、4D.Class オブジェクトの名称を格納します。クラス名の大文字・小文字は区別されます。

このプロパティは 読み取り専用 です。

.new()

▶ [履歴](#)

.new(param : any { ;...paramN }) : 4D.Class

引数	タイプ		説明
param	any	->	コンストラクター関数に渡す引数
戻り値	4D.Class	<-	クラスの新規オブジェクト

説明

.new() 関数は、対象クラスの新規インスタンスである cs.className オブジェクトを作成して返します。この関数は、cs クラスストア に属する全クラスで自動的に利用可能です。

任意の param パラメーターに渡した引数は、当該クラス定義内の Class Constructor 関数 (あれば) が受け取ります。コンストラクター関数においては、This は新規に作成されるオブジェクトを指します。

存在しないクラスを対象に .new() を呼び出した場合、エラーが返されます。

例題

Person クラスの新規インスタンスを作成するには、次のように書きます:

```
var $person : cs.Person  
$person:=cs.Person.new() // 新規インスタンスの作成  
// $person はクラス関数を格納しています
```

パラメーターを使って、Personクラスの新規インスタンスを作成するには、次のように書きます：

```
// クラス: Person.4dm  
Class constructor($firstname : Text; $lastname : Text; $age : Integer)  
    This.firstName:=$firstname  
    This.lastName:=$lastname  
    This.age:=$age
```

```
// メソッド内の使用例  
var $person : cs.Person  
$person:=cs.Person.new("John"; "Doe"; 40)  
// $person.firstName = "John"  
// $person.lastName = "Doe"  
// $person.age = 40
```

.superclass

▶ 覆歴

.superclass : 4D.Class

説明

.superclass プロパティは、対象クラスの親クラスを返します。スーパークラスは、4D.Class オブジェクト、あるいは cs.className オブジェクトのいずれかです。親クラスが存在しない場合は、このプロパティは null を返します。

ユーザークラスのスーパークラスは、`Class extends <superclass>` キーワードを使ってクラス内で定義されます。

このプロパティは 読み取り専用 です。

例題

```
$sup:=4D.File.superclass // Document  
$sup:=4D.Document.superclass // Object  
$sup:=4D.Object.superclass // null  
  
// `Class extends File` を使って  
// MyFile クラスを作成した場合  
$sup:=cs.MyFile.superclass // File
```

参照: [Super](#)

Collection

Collectionクラスはコレクション型の変数を扱います。

コレクションは次のように初期化します:

New collection <code>{(...value : any)} : Collection</code>
空の、あるいは値の入った新規コレクションを作成し、その参照を返します
New shared collection <code>{(...value : any)} : Collection</code>
空の、あるいは値が入った新規コレクションを作成し、その参照を返します

例題

```
var $colVar : Collection // コレクション型の 4D変数の宣言  
$colVar:=New collection // コレクションの初期化と 4D変数への代入
```

概要

.average(<code>{propertyPath : Text}</code>) : Real
コレクションインスタンス内で定義されている値の算術平均を返します
.clear() : Collection
コレクションインスタンス内の全要素を削除し、空のコレクションを返します
.combine(<code>col2 : Collection {; index : Integer}</code>) : Collection
コレクションインスタンスの最後、あるいは <code>index</code> で指定した位置に <code>col2</code> の要素を挿入し、変更された元のコレクションを返します
.concat(<code>value : any { ;...valueN }</code>) : Collection
<code>value</code> に指定した要素を元のコレクションの最後に追加した、新しいコレクションを返します
.copy() : Collection
.copy(<code>option : Integer</code>) : Collection
.copy(<code>option : Integer ; groupWithCol : Collection</code>) : Collection
.copy(<code>option : Integer ; groupWithObj : Object</code>) : Collection
コレクションインスタンスのディープ・コピーを返します
.count(<code>{ propertyPath : Text }</code>) : Real
コレクション内の、null ではない要素の個数を返します
.countValues(<code>value : any {; propertyPath : Text}</code>) : Real
<code>value</code> 引数に指定した値がコレクション内において見つかった回数を返します
.distinct(<code>{ option : Integer}</code>) : Collection
.distinct(<code>propertyPath : Text {; option : Integer}</code>) : Collection
元のコレクションから重複しない（異なる）値のみを格納した新しいコレクションを返します
.equal(<code>collection2 : Collection {; option : Integer}</code>) : Boolean
コレクションを <code>collection2</code> とディープ比較し、同一の場合には <code>true</code> を返します
.every(<code>methodName : Text { ;...param : any }</code>) : Boolean
.every(<code>startFrom : Integer ; methodName : Text { ;...param : any }</code>) : Boolean

コレクション内の全要素が、*methodName* に指定したメソッドで実装されたテストにパスした場合には true を返します

.extract(*propertyPath* : Text { ; *option* : Integer }) : Collection

.extract(*propertyPath* : Text ; *targetPath* : Text { ; ...*propertyPathN* : Text ; ... *targetPathN* : Text }) : Collection
元のオブジェクトのコレクションから、*propertyPath* 引数が指定するプロパティ値を抽出し、新しいコレクションに格納して返します

.fill(*value* : any) : Collection

.fill(*value* : any ; *startFrom* : Integer { ; *end* : Integer }) : Collection

コレクションを *value* 引数の値で満たし、同コレクションを返します。オプションとして、*startFrom* および *end* インデックスを渡して代入開始位置および終了位置を指定することもできます

.filter(*methodName* : Text { ; ...*param* : any }) : Collection

元のコレクション要素のうち、*methodName* メソッドの結果が true になる要素をすべて格納した新しいコレクションを返します

.find(*methodName* : Text { ; ...*param* : any }) : any

.find(*startFrom* : Integer ; *methodName* : Text { ; ...*param* : any }) : any

methodName 引数のメソッドを各コレクション要素に適用して、true を返す最初の要素を返します

.findIndex(*methodName* : Text { ; ...*param* : any }) : Integer

.findIndex(*startFrom* : Integer ; *methodName* : Text { ; ...*param* : any }) : Integer

methodName 引数のメソッドを各コレクション要素に適用して、true を返す最初の要素のインデックスを返します

.indexOf(*toSearch* : expression { ; *startFrom* : Integer }) : Integer

toSearch 引数の式をコレクション要素の中から検索し、最初に見つかった要素のインデックス（見つからなかった場合には -1）を返します

.indices(*queryString* : Text { ; ...*value* : any }) : Collection

queryString 引数の検索条件に合致する、元のコレクション要素のインデックスを返します

.insert(*index* : Integer ; *element* : any) : Collection

index で指定したコレクションインスタンスの位置に *element* 要素を挿入し、変更された元のコレクションを返します

.join(*delimiter* : Text { ; *option* : Integer }) : Text

delimiter に渡した文字列を区切り文字として、コレクションの全要素を一つの文字列につなげます

.lastIndexOf(*toSearch* : expression { ; *startFrom* : Integer }) : Integer

toSearch 引数の式をコレクション要素の中から検索し、最後に見つかった要素のインデックス（見つからなかった場合には -1）を返します

[.length : Integer

](#length)

コレクション内の要素数を返します

.map(*methodName* : Text { ; ...*param* : any }) : Collection

元のコレクションの各要素に対して *methodName* メソッドを呼び出した結果に基づいた、新しいコレクションを作成します

.max({ *propertyPath* : Text }) : any

コレクション内の最大値を持つ要素を返します

.min({ *propertyPath* : Text }) : any

コレクション内の最小値を持つ要素を返します

.orderBy() : Collection

.orderBy(*pathStrings* : Text) : Collection

.orderBy(*pathObjects* : Collection) : Collection

.orderBy(*ascOrDesc* : Integer) : Collection

コレクションの要素を指定順に並べ替えた新しいコレクションを返します

.orderByMethod(*methodName* : Text { ; ...*extraParam* : expression }) : Collection

methodName メソッドを通して定義された順番でコレクション要素を並べ替えた新しいコレクションを返します

<code>.pop() : any</code>	コレクションから最後の要素を取り除き、それを戻り値として返します
<code>.push(element : any { ;...elementN }) : Collection</code>	一つ以上の <code>element</code> 引数をコレクションインスタンスの最後に追加し、変更された元のコレクションを返します
<code>.query(queryString : Text ; ...value : any) : Collection</code>	
<code>.query(queryString : Text ; querySettings : Object) : Collection</code>	
	検索条件に合致するオブジェクトコレクションの要素をすべて返します
<code>.reduce(methodName : Text) : any</code>	
<code>.reduce(methodName : Text ; initialValue : any { ; ...param : expression }) : any</code>	<code>methodName</code> コールバックメソッドをアキュムレーターおよびコレクションの各要素に（左から右へ）適用して、単一の値にまとめます
<code>.remove(index : Integer { ; howMany : Integer }) : Collection</code>	<code>index</code> で指定した位置から一つまたは複数のコレクション要素を削除し、変更されたコレクションを返します
<code>.resize(size : Integer { ; defaultValue : any }) : Collection</code>	コレクションの <code>length</code> を引数で指定されたサイズに設定し、変更された元のコレクションを返します
<code>.reverse() : Collection</code>	全要素が逆順になった、コレクションのディープ・コピーを返します
<code>.shift() : any</code>	コレクションの先頭要素を取り除き、それを戻り値として返します
<code>.slice(startFrom : Integer { ; end : Integer }) : Collection</code>	コレクションの一部を、新しいコレクションの中に返します
<code>.some(methodName : Text { ; ...param : any }) : Boolean</code>	
<code>.some(startFrom : Integer ; methodName : Text { ; ...param : any }) : Boolean</code>	少なくとも一つのコレクション要素が、 <code>methodName</code> に指定したメソッドで実装されたテストにパスした場合に <code>true</code> を返します
<code>.sort(methodName : Text { ; ...extraParam : any }) : Collection</code>	コレクションの要素を並べ替えます
<code>.sum({ propertyPath : Text }) : Real</code>	コレクションインスタンスの全要素の値を合計して返します
<code>.unshift(value : any { ;...valueN : any }) : Collection</code>	一つ以上の <code>value</code> 引数をコレクションインスタンスの先頭に挿入し、変更された元のコレクションを返します

New collection

New collection `{(...value : any)} : Collection`

引数	タイプ	説明
<code>value</code>	Number, Text, Date, Time, Boolean, Object, Collection, Picture, Pointer	-> コレクションの値
戻り値	Collection	<- 新しいコレクション

説明

`New collection` コマンドは、空の、あるいは値の入った新規コレクションを作成し、その参照を返します。

引数を渡さなかった場合、`New collection` は空のコレクションを作成し、その参照を返します。

返された参照は、コレクション型の 4D変数に代入する必要があります。

`var : Collection` や `C_COLLECTION` ステートメントはコレクション型の変数を宣言しますが、コレクション自体は作成しないという点に注意してください。

任意で、一つ以上の `value` 引数を渡すことで、あらかじめ値の入った新しいコレクションを作成することができます。

または、あとから代入によって要素を一つずつ追加・編集していくことができます。たとえば：

```
myCol[10] := "My new element"
```

コレクションの最終要素を超える要素番号（インデックス）を指定した場合、コレクションは自動的にリサイズされ、合い間の要素にはすべて `null` 値が割り当てられます。

サポートされている型（数値、テキスト、日付、ピクチャー、ポインター、オブジェクト、コレクション等）であれば、個数に制限なく値を渡すことができます。配列とは異なり、コレクションでは異なる型のデータを混ぜることができます。

ただし以下の変換問題については注意する必要があります：

- 渡されたポインターは、そのまま保存されます。ポインターは `JSON Stringify` コマンドを使用することで評価されます。
- 日付は、"dates inside objects" データベース設定に応じて、"yyyy-mm-dd" という日付、または "YYYY-MM-DDTHH:mm:ss.SSSZ" というフォーマットの文字列で保存されます。コレクションに保存する前に 4D日付をテキストに変換した場合、プログラムはデフォルトでローカルのタイムゾーンを使用します。このふるまいは `SET DATABASE PARAMETER` コマンドで `Dates inside objects` セレクターを使用することで変更可能です。
- 時間を渡した場合、それはミリ秒の数（実数）として保存されます。

例題 1

新しい空のコレクションを作成し、それを 4Dコレクション変数に代入します：

```
var $myCol : Collection  
$myCol:=New collection  
// $myCol=[]
```

例題 2

あらかじめ値の入ったコレクションを作成します：

```
var $filledColl : Collection  
$filledColl:=New collection(33;"mike";"november";->myPtr;Current date)  
// $filledColl=[33,"mike","november","->myPtr","2017-03-28T22:00:00.000Z"]
```

例題 3

新しいコレクションを作成し、そこに新しい要素を追加します：

```
var $coll : Collection  
$coll:=New collection("a";"b";"c")  
// $coll=["a","b","c"]  
$coll[9]:="z" // 値 "z" を10番目の要素として追加します  
$collSize:=$coll.length // 10  
// $coll=["a","b","c",null,null,null,null,null,"z"]
```

New shared collection

▶ 履歴

New shared collection {(...value : any)} : Collection

引数	タイプ	説明
value	Number, Text, Date, Time, Boolean, Shared object, Shared collection	-> 共有コレクションの値
戻り値	Collection	<- 新規の共有コレクション

説明

New shared collection コマンドは、空の、あるいは値が入った新規コレクションを作成し、その参照を返します。

このコレクションに要素を追加する場合には `Use...End use` 構造でくる必要があります。そうしない場合にはエラーが返されます。ただし、属性の読み取りは `Use...End use` 構造の外側でも可能です。

共有コレクションについての詳細は、[共有オブジェクトと共有コレクション](#) のページを参照してください。

引数を渡さない場合、New shared collection は空のコレクションを作成し、その参照を返します。

返された参照は、コレクション型の 4D変数に代入する必要があります。

`var : Collection` や `C_COLLECTION` ステートメントはコレクション型の変数を宣言しますが、コレクション自体は作成しないという点に注意してください。

任意で、一つ以上の value 引数を渡すことで、あらかじめ値の入った新しい共有コレクションを作成することができます。または、あとからオブジェクト記法による代入で要素を一つずつ追加・編集していくことができます（例題参照）。

共有コレクションの最終要素を超える要素番号（インデックス）を指定した場合、共有コレクションは自動的にリサイズされ、合い間の要素にはすべて null 値が割り当てられます。

以下のサポートされる型であれば、いくつでも値を渡すことができます：

- 数値（実数、倍長整数…）。数値は常に実数として保存されます。
- テキスト
- ブール
- 日付
- 時間（ミリ秒の数（実数）として保存されます）。
- null
- 共有オブジェクト(*)
- 共有コレクション(*)

標準のコレクション（非共有コレクション）とは異なり、共有コレクションはピクチャーやポインター、共有でないオブジェクトおよびコレクションはサポートしていません。

（）共有オブジェクトおよびコレクションが共有コレクションに追加された場合、それらは同じロック識別子を共有します。この点についてのより詳細は、4D ランゲージリファレンス* の [ロック識別子](#) の章を参照してください。

例題

```
$mySharedCol:=New shared collection("alpha";"omega")
Use($mySharedCol)
  $mySharedCol[1]:="beta"
End use
```

.average()

▶ 履歴

`.average({propertyPath : Text}) : Real`

引数	タイプ		説明
propertyPath	Text	->	計算に使用するオブジェクトプロパティのパス
戻り値	Real, Undefined	<-	コレクションの値の算術平均

説明

`.average()` 関数は、コレクションインスタンス内で定義されている値の算術平均を返します。

計算の対象となるのは数値のみです（他の型の要素は無視されます）。

コレクションがオブジェクトを格納している場合には、計算するオブジェクトプロパティのパスを `propertyPath` に渡します。

`.average()` は以下の場合には `undefined` を返します：

- コレクションが空の場合
- コレクションに数値が含まれていない場合
- `propertyPath` 引数で指定したパスがコレクション内で見つからない場合

例題 1

```
var $col : Collection
$col:=New collection(10;20;"Monday";True;6)
$vAvg:=$col.average() //12
```

例題 2

```
var $col : Collection
$col:=New collection
$col.push(New object("name";"Smith";"salary";10000))
$col.push(New object("name";"Wesson";"salary";50000))
$col.push(New object("name";"Gross";"salary";10500))
$vAvg:=$col.average("salary") //23500
```

.clear()

▶履歴

`.clear() : Collection`

引数	タイプ		説明
戻り値	Collection	<-	全要素が削除された元のコレクション

説明

`.clear()` 関数は、コレクションインスタンス内の全要素を削除し、空のコレクションを返します。

このコマンドは、元のコレクションを変更します。

例題

```

var $col : Collection
$col:=New collection(1;2;5)
$col.clear()
$vSize:=$col.length // $vSize=0

```

.combine()

▶履歴

.combine(col2 : Collection {; index : Integer }) : Collection

引数	タイプ		説明
col2	Collection	->	追加するコレクション
index	Integer	->	追加要素を挿入する位置 (デフォルトは length+1)
戻り値	Collection	<-	追加要素を格納した元のコレクション

説明

.combine() 関数は、コレクションインスタンスの最後、あるいは index で指定した位置に col2 の要素を挿入し、変更された元のコレクションを返します。.insert() 関数とは異なり、.combine() は col2 の各要素を元のコレクション追加します (col2 自体を単一のコレクション要素としては挿入しません)。

このコマンドは、元のコレクションを変更します。

デフォルトでは、col2 の要素は元のコレクションの最後に追加されます。index に引数を渡すことで、col2 の要素を挿入する位置を指定することができます。

警告: コレクション要素は 0 起点である点に注意してください。

- 指定した index がコレクションの length より大きい場合、実際の開始インデックスはコレクションの length に設定されます。
- index < 0 の場合、index:=index+length として再計算されます (コレクションの終端からのオフセットであるとみなされます)。
- 計算結果も負の値である場合、index は 0 に設定されます。

例題

```

var $c; $fruits : Collection
$c:=New collection(1;2;3;4;5;6)
$fruits:=New collection("Orange";"Banana";"Apple";"Grape")
$c.combine($fruits;3) // [1,2,3,"Orange","Banana","Apple","Grape",4,5,6]

```

.concat()

▶履歴

.concat(value : any { ;...valueN }) : Collection

引数	タイプ		説明
value	Number, Text, Object, Collection, Date, Time, Boolean, Picture	->	連結する値。value がコレクションの場合、コレクションの全要素が元のコレクションに追加されます。
戻り値	Collection	<-	元のコレクションに値が追加された新規コレクション

説明

`.concat()` 関数は、`value` に指定した要素を元のコレクションの最後に追加した、新しいコレクションを返します。

このコマンドは、元のコレクションを変更しません。

`value` がコレクションの場合、その全要素が新しい要素として元のコレクションの最後に追加されます。 `value` がコレクションでない場合、それ自体が新しい要素として追加されます。

例題

```
var $c : Collection
$c:=New collection(1;2;3;4;5)
$fruits:=New collection("Orange";"Banana";"Apple";"Grape")
$fruits.push(New object("Intruder";"Tomato"))
$c2:=$c.concat($fruits) // [1,2,3,4,5,"Orange","Banana","Apple","Grape",{"Intruder":"Tomato"}]
$c2:=$c.concat(6;7;8) // [1,2,3,4,5,6,7,8]
```

.copy()

▶履歴

`.copy()` : Collection
`.copy(option : Integer)` : Collection
`.copy(option : Integer ; groupWithCol : Collection)` : Collection
`.copy(option : Integer ; groupWithObj : Object)` : Collection

引数	タイプ	説明
option	Integer	-> ck resolve pointers : コピー前にポインターを解決する ck shared : 共有コレクションを返す
groupWithCol	Collection	-> 結果のコレクションとグループする共有コレクション
groupWithObj	Object	-> 結果のコレクションとグループする共有オブジェクト
戻り値	Collection	<- 元のコレクションのディープ・コピー

説明

`.copy()` 関数は、コレクションインスタンスのディープ・コピーを返します。ディープ・コピーとは、元のコレクション内のオブジェクトやコレクションは複製されることを意味し、返されたコレクションと元のコレクションは参照を共有しないということを意味します。

このコマンドは、元のコレクションを変更しません。

任意の `option` パラメーターには、以下のどちらか（あるいは両方）の定数を渡すことができます：

option	説明
ck resolve pointers	オリジナルのコレクションがポインター型の値を格納している場合、デフォルトではコピー先のオブジェクトもポインターを格納します。しかししながら、 <code>ck resolve pointers</code> 定数を渡すことで、コピー時にポインターを解決することができます。この場合、コレクション内の各ポインターはコピー時に解決され、解決済みの値が使用されます。
ck shared	共有コレクションに対して適用された場合でも、 <code>copy()</code> はデフォルトで通常の（非共有の）コレクションを返します。共有コレクションを作成するには、 <code>ck shared</code> 定数を渡します。この場合には、 <code>groupWith</code> パラメーターに引数を渡して他の共有オブジェクトまたは共有コレクションに関連づけることもできます（以下参照）。

`groupWithCol` または `groupWithObj` 引数を渡すと、結果のコレクションを関連づけるコレクションまたはオブジェクトを指定できます。

例題 1

通常の（非共有の）コレクション `$lastnames` * を、共有オブジェクト `$sharedObject` 内にコピーします。このためには、まず共有コレクション `($sharedLastnames*)` を作成する必要があります。

```

var $sharedObject : Object
var $lastnames;$sharedLastnames : Collection
var $text : Text

$sharedObject:=New shared object

$text:=Document to text(Get 4D folder(Current resources folder)+"lastnames.txt")
$lastnames:=JSON Parse($text) // $lastnames は通常のコレクションです

$sharedLastnames:=$lastnames.copy(ck shared) // $sharedLastnames は共有コレクションです

// $sharedLastnames は $sharedObject の中に入れられます
Use($sharedObject)
  $sharedObject.lastnames:=$sharedLastnames
End use

```

例題 2

どちらも共有コレクションである \$sharedColl1 と \$sharedColl2 を結合します。これらは異なる共有グループに所属しているため、直接結合した場合にはエラーが生成されます。そこで、\$sharedColl1 のコピーを作成し、\$sharedColl2 をそのコピーの共有グループ先に指定します。

```

var $sharedColl1;$sharedColl2;$copyColl : Collection

$sharedColl1:=New shared collection(New shared object("lastname";"Smith"))
$sharedColl2:=New shared collection(New shared object("lastname";"Brown"))

// $copyColl を $sharedColl2 と同じ共有グループに所属させます
$copColl:=$sharedColl1.copy(ck shared;$sharedColl2)
Use($sharedColl2)
  $sharedColl2.combine($copyColl)
End use

```

例題 3

通常のコレクション (\$lastnames) があり、それをアプリケーションの Storage に入れます。これには、先に共有コレクション（\$sharedLastnames）を作成しておく必要があります。

```

var $lastnames;$sharedLastnames : Collection
var $text : Text

$text:=Document to text(Get 4D folder(Current resources folder)+"lastnames.txt")
$lastnames:=JSON Parse($text) // $lastnames は通常の（非共有）コレクションです

$sharedLastnames:=$lastnames.copy(ck shared) // 共有コピー

Use(Storage)
  Storage.lastnames:=$sharedLastnames
End use

```

例題 4

`ck resolve pointers` オプションを使用した場合のふるまいです:

```

var $col : Collection
var $p : Pointer
$p:==>$what

$col:=New collection
$col.push(New object("alpha";"Hello";"num";1))
$col.push(New object("beta";"You";"what";$p))

$col2:=$col.copy()
$col2[1].beta:="World!"
ALERT($col[0].alpha+" "+$col2[1].beta) // "Hello world!" を表示します

$what:="You!"
$col3:=$col2.copy(ck resolve pointers)
ALERT($col3[0].alpha+" "+$col3[1].what) // "Hello You!" を表示します

```

.count()

▶履歴

.count({ *propertyPath* : Text }) : Real

引数	タイプ	説明
<i>propertyPath</i>	Text	-> 計算に使用するオブジェクトプロパティのパス
戻り値	Real	<- コレクション内の要素の数

説明

.count() 関数は、コレクション内の、null ではない要素の個数を返します。

コレクションがオブジェクトを含んでいる場合、*propertyPath* 引数を渡すことができます。この場合、*propertyPath* で指定したパスを含む要素のみがカウントされます。

例題

```

var $col : Collection
var $count1;$count2 : Real
$col:=New collection(20;30;Null;40)
$col.push(New object("name";"Smith";"salary";10000))
$col.push(New object("name";"Wesson";"salary";50000))
$col.push(New object("name";"Gross";"salary";10500))
$col.push(New object("lastName";"Henry";"salary";12000))
$count1:=$col.count() // $count1=7
$count2:=$col.count("name") // $count2=3

```

.countValues()

▶履歴

.countValues(*value* : any {; *propertyPath* : Text }) : Real

引数	タイプ	説明
<i>value</i>	Text, Number, Boolean, Date, Object, Collection	-> 数える値
<i>propertyPath</i>	Text	-> 計算に使用するオブジェクトプロパティのパス
戻り値	Real	<- 値の出現回数

説明

`.countValues()` 関数は、`value` 引数に指定した値がコレクション内において見つかった回数を返します。

`value` には、以下のいずれかを渡すことができます：

- スカラー値（テキスト、数値、ブール、日付）
- オブジェクトあるいはコレクションの参照

要素が検出されるためには、`value` 引数の型が要素の型と合致している必要があります。このファンクションは等号演算子を使用します。

任意の `propertyPath` 引数を渡すと、オブジェクトのコレクションにおける値の個数を数えることができます。`propertyPath` には値を検索するプロパティパスを渡します。

このコマンドは、元のコレクションを変更しません。

例題 1

```
var $col : Collection
var $vCount : Integer
$col:=New collection(1;2;5;5;5;3;6;4)
$vCount:=$col.countValues(5) // $vCount=3
```

例題 2

```
var $col : Collection
var $vCount : Integer
$col:=New collection
$col.push(New object("name";"Smith";"age";5))
$col.push(New object("name";"Wesson";"age";2))
$col.push(New object("name";"Jones";"age";3))
$col.push(New object("name";"Henry";"age";4))
$col.push(New object("name";"Gross";"age";5))
$vCount:=$col.countValues(5;"age") // $vCount=2
```

例題 3

```
var $numbers; $letters : Collection
var $vCount : Integer

$letters:=New collection("a";"b";"c")
$numbers:=New collection(1;2;$letters;3;4;5)

$vCount:=$numbers.countValues($letters) // $vCount=1
```

.distinct()

▶履歴

`.distinct({ option : Integer }) : Collection`

`.distinct(propertyPath : Text {; option : Integer }) : Collection`

引数	タイプ		説明
option	Integer	->	<code>ck diacritical</code> : アクセント等の発音区別符号を無視しない評価（たとえば "A" # "a"）
propertyPath	Text	->	重複しない値を取得する属性のパス
戻り値	Collection	<-	重複しない値のみを格納した新規コレクション

説明

.**distinct()** 関数は、元のコレクションから重複しない（異なる）値のみを格納した新しいコレクションを返します。

このコマンドは、元のコレクションを変更しません。

返されたコレクションは自動的に並べ替えられています。 Null 値は返されません。

デフォルトでは、アクセント等の発音区別符号を無視した評価が実行されます。評価の際に文字の大小を区別したり、アクセント記号を区別したい場合には、option に `ck diacritical` 定数を渡します。

コレクションがオブジェクトを格納している場合には、重複しない値を取得するオブジェクトプロパティのパスを `propertyName` に渡します。

例題

```
var $c; $c2 : Collection
$c:=New collection
$c.push("a";"b";"c";"A";"B";"c";"b";"b")
$c.push(New object("size";1))
$c.push(New object("size";3))
$c.push(New object("size";1))
$c2:=$c.distinct() // $c2=[{"a": "a", "b": "b", "c": "c", "size": 1}, {"a": "A", "b": "B", "c": "c", "size": 3}, {"a": "b", "b": "b", "c": "c", "size": 1}]
$c2:=$c.distinct(ck diacritical) // $c2=[{"a": "a", "b": "b", "c": "c", "size": 1}, {"a": "A", "b": "B", "c": "c", "size": 3}, {"a": "b", "b": "b", "c": "c", "size": 1}]
$c2:=$c.distinct("size") // $c2=[1, 3]
```

.equal()

▶履歴

.**equal(collection2 : Collection {; option : Integer })** : Boolean

引数	タイプ		説明
collection2	Collection	->	比較するコレクション
option	Integer	->	<code>ck diacritical</code> : アクセント等の発音区別符号を無視しない評価（たとえば "A" # "a"）
戻り値	Boolean	<-	コレクションが同一の場合には true、それ以外は false

説明

.**equal()** 関数は、コレクションを `collection2` とディープ比較し、同一の場合には `true` を返します。

デフォルトでは、アクセント等の発音区別符号を無視した評価が実行されます。評価の際に文字の大小を区別したり、アクセント記号を区別したい場合には、option に `ck diacritical` 定数を渡します。

Null値の要素は undefined要素と同じとはみなされません。

例題

```

var $c; $c2 : Collection
var $b : Boolean

$c:=New collection(New object("a";1;"b";"orange");2;3)
$c2:=New collection(New object("a";1;"b";"orange");2;3;4)
$b:=$c.equal($c2) // false

$c:=New collection(New object("1";"a";"b";"orange");2;3)
$c2:=New collection(New object("a";1;"b";"orange");2;3)
$b:=$c.equal($c2) // false

$c:=New collection(New object("a";1;"b";"orange");2;3)
$c2:=New collection(New object("a";1;"b";"0Range");2;3)
$b:=$c.equal($c2) // true

$c:=New collection(New object("a";1;"b";"orange");2;3)
$c2:=New collection(New object("a";1;"b";"0Range");2;3)
$b:=$c.equal($c2);ck diacritical) //false

```

.every()

▶履歴

`.every(methodName : Text { ;...param : any }) : Boolean`
`.every(startFrom : Integer ; methodName : Text { ;...param : any }) : Boolean`

引数	タイプ		説明
startFrom	Integer	->	テストを開始するインデックス
methodName	Text	->	テストに呼び出すメソッド名
param	Mixed	->	methodName に渡す引数
戻り値	Boolean	<-	すべての要素がテストをパスすれば true

説明

`.every()` 関数は、コレクション内の全要素が、*methodName* に指定したメソッドで実装されたテストにパスした場合には `true` を返します。

methodName には、コレクション要素の評価に使用するメソッド名を渡します。*param* には、必要に応じて引数を渡します（任意）。*methodName* で指定したメソッドはどんなテストでも実行でき、引数はあっても構いません。このメソッドは `$1` にオブジェクトを受け取り、テストをパスした要素の `$1.result` を `true` に設定しなければなりません。

methodName で指定したメソッドは以下の引数を受け取ります：

- `$1.value`: 評価する要素の値
- `$2: param`
- `$N...: paramN...`

methodName で指定したメソッドでは、以下の引数を設定します：

- `$1.result` (布尔): 要素の値の評価が成功した場合には `true`、それ以外は `false`
- `$1.stop` (布尔、任意): メソッドコールバックを止める場合には `true`。返された値は最後に計算されたものです。

`.every()` 関数は、`$1.result` に `false` を返すコレクション要素を見つけると、*methodName* メソッドの呼び出しをやめて `false` を返します。

デフォルトでは、`.every()` はコレクション全体をテストします。任意で、*startFrom* にテストを開始する要素のインデックスを渡すこともできます。

- *startFrom* がコレクションの `length` 以上だった場合、`false` が返されます。これはコレクションがテストされていないことを意味します。
- *startFrom* < 0 の場合には、コレクションの終わりからのオフセットであるとみなされます(`startFrom:=startFrom+length`)。
- *startFrom* = 0 の場合、コレクション全体がテストされます（デフォルト）。

例題 1

```

var $c : Collection
var $b : Boolean
$c:=New collection
$c.push(5;3;1;4;6;2)
$b:=$c.every("NumberGreaterThan0") // true を返します
$c.push(-1)
$b:=$c.every("NumberGreaterThan0") // false を返します

```

NumberGreaterThan0 メソッドの中身は以下のとおりです:

```
$1.result:=$1.value>0
```

例題 2

コレクション要素がすべて実数型であるかをテストします:

```

var $c : Collection
var $b : Boolean
$c:=New collection
$c.push(5;3;1;4;6;2)
$b:=$c.every("TypeLookUp";Is real) // $b=true
$c:=$c.push(New object("name";"Cleveland";"zc";35049))
$c:=$c.push(New object("name";"Blountsville";"zc";35031))
$b:=$c.every("TypeLookUp";Is real) // $b=false

```

TypeLookUp メソッドの中身は以下のとおりです:

```

#DECLARE ($toEval : Object ; $param : Integer) // $1; $2
If(Value type($toEval.value)=$param)
    $toEval.result:=True
End if

```

.extract()

▶ 覆歴

.extract(propertyPath : Text { ; option : Integer }) : Collection

.extract(propertyPath : Text ; targetPath : Text { ;...propertyPathN : Text ;... targetPathN : Text }) : Collection

引数	タイプ		説明
propertyPath	Text	->	新しいコレクションに抽出する値のオブジェクトプロパティパス
targetpath	Text	->	抽出先のプロパティパスあるいはプロパティ名
option	Integer	->	ck keep null : 返されるコレクションに null プロパティを含めます (デフォルトでは無視されます)。 <i>targetPath</i> を渡した場合には、この引数は無視されます。
戻り値	Collection	<-	抽出した値を格納した新しいコレクション

説明

.extract() 関数は、元のオブジェクトのコレクションから、*propertyPath* 引数が指定するプロパティ値を抽出し、新しいコレクションに格納して返します。

このコマンドは、元のコレクションを変更しません。

戻り値のコレクションの中身は、*targetPath* 引数によります:

- *targetPath* が省略された場合、`.extract()` は元のコレクションの *propertyPath* と同じパスを使って、新しいコレクションに値を格納します。
デフォルトでは、*propertyPath* のパスの要素が `null` あるいは `undefined` であった場合には、その要素は無視され、返されるコレクションに格納されません。*option* パラメーターに `ck keep null` 定数を渡すと、これらの要素は返されるコレクションに `null` 要素として格納されます。
- 一つ以上の *targetPath* 引数が渡された場合、`.extract()` は元のコレクションの *propertyPath* から値を抽出し、対応する *targetPath* に値を保存したオブジェクトを新しいコレクションの各要素として格納します。Null値はそのまま保持されます（このシンタックスでは *option* に引数を渡しても無視されます）。

例題 1

```
var $c : Collection
$c:=New collection
$c.push(New object("name";"Cleveland"))
$c.push(New object("zip";5321))
$c.push(New object("name";"Blountsville"))
$c.push(42)
$c2:=$c.extract("name") // $c2=[Cleveland,Blountsville]
$c2:=$c.extract("name";ck keep null) // $c2=[Cleveland,null,Blountsville,null]
```

例題 2

```
var $c : Collection
$c:=New collection
$c.push(New object("zc";35060))
$c.push(New object("name";Null;"zc";35049))
$c.push(New object("name";"Cleveland";"zc";35049))
$c.push(New object("name";"Blountsville";"zc";35031))
$c.push(New object("name";"Adger";"zc";35006))
$c.push(New object("name";"Clanton";"zc";35046))
$c.push(New object("name";"Clanton";"zc";35045))
$c2:=$c.extract("name";"City") // $c2=[{City:null},{City:Cleveland},{City:Blountsville},{City:Adger},{City:Clanton},{City:Clanton}]
$c2:=$c.extract("name";"City";"zc";"Zip") // $c2=[{Zip:35060},{City:null,Zip:35049},{City:Cleveland,Zip:35031}]
```

.fill()

▶ 覆歴

`.fill(value : any) : Collection`
`.fill(value : any ; startFrom : Integer { ; end : Integer }) : Collection`

引数	タイプ		説明
value	number, Text, Collection, Object, Date, Boolean	->	代入する値
startFrom	Integer	->	開始インデックス（含まれる）
end	Integer	->	終了インデックス（含まれない）
戻り値	collection	<-	値が代入された元のコレクション

説明

`.fill()` 関数は、コレクションを *value* 引数の値で満たし、同コレクションを返します。オプションとして、*startFrom* および *end* インデックスを渡して代入開始位置および終了位置を指定することもできます。

このコマンドは、元のコレクションを変更します。

- *startFrom* 引数が渡されなかった場合、*value* 引数の値はコレクションの全要素に代入されます (つまり、*startFrom*=0)。
- *startFrom* 引数が渡され、かつ *end* 引数が省略された場合には、*value* 引数の値はコレクションの最後の要素まで設定されます (つまり、*end*=*length*)。
- *startFrom* と *end* 引数が両方渡された場合には、*startFrom* から *end* までの要素に *value* が代入されます。

引数に矛盾がある場合、次のように解釈されます:

- *startFrom* < 0 の場合、*startFrom*:=*startFrom*+*length* として再計算されます (コレクションの終端からのオフセットであるとみなされます)。再計算された値も負の値だった場合、*startFrom* は 0 に設定されます。
- *end* < 0 の場合、それは *end*:=*end*+*length* として再計算されます。
- 渡された値、あるいは再計算された値が *end* < *startFrom* の場合、関数はなにもしません。

例題

```
var $c : Collection
$c:=New collection(1;2;3;"Lemon";Null;"";4;5)
$c.fill("2") // $c:=[2,2,2,2,2,2,2]
$c.fill("Hello";5) // $c=[2,2,2,2,Hello,Hello,Hello]
$c.fill(0;1;5) // $c=[2,0,0,0,0>Hello,Hello,Hello]
$c.fill("world";1;-5) // -5+8=3 -> $c=[2,"world","world",0,0>Hello,Hello,Hello]
```

.filter()

▶ 覆歴

.filter(*methodName* : Text { ; ...*param* : any }) : Collection

引数	タイプ		説明
<i>methodName</i>	Text	->	コレクションをフィルターするために呼び出すメソッド名
<i>param</i>	Mixed	->	<i>methodName</i> に渡す引数
戻り値	Collection	<-	フィルターされた要素を格納した新しいコレクション(シャロウ・コピー)

説明

.filter() 関数は、元のコレクション要素のうち、*methodName* メソッドの結果が true になる要素をすべて格納した新しいコレクションを返します。この関数は シャロウ・コピー を返します。つまり、元のコレクションにオブジェクト要素やコレクション要素が含まれていた場合、それらの参照は戻り値のコレクションで共有されます。また、元のコレクションが共有コレクションであった場合、返されるコレクションもまた共有コレクションになります。

このコマンドは、元のコレクションを変更しません。

methodName には、コレクション要素の評価に使用するメソッド名を渡します。*param* には、必要に応じて引数を渡します(任意)。*methodName* で指定したメソッドはどんなテストでも実行でき、引数はあっても構いません。このメソッドは \$1 にオブジェクトを受け取り、メソッドの条件を満たして新規コレクションに代入されるべき要素の \$1.result を true に設定しなければなりません。

methodName で指定したメソッドは以下の引数を受け取ります:

- \$1.value: フィルターする要素の値
- \$2: *param*
- \$N...: param2...paramN

methodName で指定したメソッドでは、以下の引数を設定します:

- \$1.result (布尔): 要素の値がフィルターの条件に合致し、新コレクションに代入すべき場合に true
- \$1.stop (布尔、任意): メソッドコールバックを止める場合には true。返された値は最後に計算されたものです。

例題 1

コレクションから、長さが 6未満であるテキスト要素を取得します:

```

var $col;$colNew : Collection
$col:=New collection("hello";"world";"red horse";66;"tim";"san jose";"miami")
$colNew:=$col.filter("LengthLessThan";6)
// $colNew=["hello","world","tim","miami"]

```

LengthLessThan メソッドのコードは以下のとおりです:

```

C_OBJECT($1)
C_LONGINT($2)
If(Value type($1.value)=Is text)
    $1.result:=(Length($1.value))<$2
End if

```

例題 2

値の型に応じて要素をフィルターします:

```

var $c;$c2;$c3 : Collection
$c:=New collection(5;3;1;4;6;2)
$c.push(New object("name";"Cleveland";"zc";35049))
$c.push(New object("name";"Blountsville";"zc";35031))
$c2:=$c.filter("TypeLookUp";Is real) // $c2=[5,3,1,4,6,2]
$c3:=$c.filter("TypeLookUp";Is object)
// $c3=[{name:Cleveland,zc:35049},{name:Blountsville,zc:35031}]

```

TypeLookUp メソッドのコードは以下のとおりです:

```

C_OBJECT($1)
C_LONGINT($2)
If(0B Get type($1;"value")=$2)

    $1.result:=True
End if

```

.find()

▶ 履歴

```

.find( methodName : Text { ; ...param : any } ) : any
.find( startFrom : Integer ; methodName : Text { ; ...param : any } ) : any

```

引数	タイプ		説明
startFrom	Integer	->	検索を開始するインデックス
methodName	Text	->	検索用に呼び出すメソッド名
param	any	->	methodName に渡す引数
戻り値	any	<-	最初に見つかった値。見つからなかった場合には Undefined

説明

.find() 関数は、*methodName* 引数のメソッドを各コレクション要素に適用して、true を返す最初の要素を返します。

このコマンドは、元のコレクションを変更しません。

methodName には、コレクション要素の評価に使用するメソッド名を渡します。*param* には、必要に応じて引数を渡します（任意）。

methodName で指定したメソッドはどんなテストでも実行でき、引数はあっても構いません。このメソッドは \$1 にオブジェクトを受け取り、条件を満たす最初の要素の \$1.result を true に設定しなければなりません。

methodName で指定したメソッドは以下の引数を受け取ります:

- \$1.value: 評価する要素の値
- in \$2: param
- \$N...: param2...paramN

methodName で指定したメソッドでは、以下の引数を設定します:

- \$1.result (布尔): 要素の値が検索条件に合致する場合に true
- \$1.stop (布尔、任意): メソッドコールバックを止める場合には true。返された値は最後に計算されたものです。

デフォルトでは、`.find()` はコレクション全体をテストします。任意で、`startFrom` に検索を開始する要素のインデックスを渡すこともできます。

- `startFrom` がコレクションの `length` 以上だった場合、-1 が返されます。これはコレクションが検索されていないことを意味します。
- `startFrom < 0` の場合には、コレクションの終わりからのオフセットであるとみなされます (`startFrom:=startFrom+length`)。注: `startFrom` が負の値であっても、コレクションは左から右へと検索されます。
- `startFrom = 0` の場合、コレクション全体がテストされます (デフォルト)。

例題 1

長さが 5未満の最初のテキスト要素を取得します:

```
var $col : Collection
$col:=New collection("hello";"world";4;"red horse";"tim";"san jose")
$value:=$col.find("LengthLessThan";5) //$/value="tim"
```

`LengthLessThan` メソッドのコードは以下のとおりです:

```
var $1 : Object
var $2 : Integer
If(Value type($1.value)=Is text)
    $1.result:=(Length($1.value))<$2
End if
```

例題 2

コレクション内を都市名で検索します:

```
var $c : Collection
var $c2 : Object
$c:=New collection
$c.push(New object("name";"Cleveland";"zc";35049))
$c.push(New object("name";"Blountsville";"zc";35031))
$c.push(New object("name";"Adger";"zc";35006))
$c.push(New object("name";"Clanton";"zc";35046))
$c.push(New object("name";"Clanton";"zc";35045))
$c2:=$c.find("FindCity";"Clanton") //$/c2={name:Clanton,zc:35046}
```

`FindCity` メソッドのコードは以下のとおりです:

```
var $1 : Object
var $2 : Text
$1.result:=$1.value.name==$2 // name は、コレクションのオブジェクト要素内のプロパティ名です
```

.findIndex()

▶ 履歴

.findIndex(*methodName* : Text { ; ...*param* : any }) : Integer

.findIndex(*startFrom* : Integer ; *methodName* : Text { ; ...*param* : any }) : Integer

引数	タイプ		説明
<i>startFrom</i>	Integer	->	検索を開始するインデックス
<i>methodName</i>	Text	->	検索用に呼び出すメソッド名
<i>param</i>	any	->	<i>methodName</i> に渡す引数
戻り値	Integer	<-	最初に見つかった値のインデックス。見つからなかった場合には -1

説明

.findIndex() 関数は、*methodName* 引数のメソッドを各コレクション要素に適用して、true を返す最初の要素のインデックスを返します。

このコマンドは、元のコレクションを変更しません。

methodName には、コレクション要素の評価に使用するメソッド名を渡します。*param* には、必要に応じて引数を渡します（任意）。*methodName* で指定したメソッドはどんなテストでも実行でき、引数はあっても構いません。このメソッドは \$1 にオブジェクトを受け取り、条件を満たす最初の要素の \$1.result を true に設定しなければなりません。

methodName で指定したメソッドは以下の引数を受け取ります：

- \$1.value: 評価する要素の値
- in \$2: *param*
- \$N...: param2...paramN

methodName で指定したメソッドでは、以下の引数を設定します：

- \$1.result (布尔): 要素の値が検索条件に合致する場合に true
- \$1.stop (布尔、任意): メソッドコールバックを止める場合には true。返された値は最後に計算されたものです。

デフォルトでは、.findIndex() はコレクション全体をテストします。任意で、*startFrom* に検索を開始する要素のインデックスを渡すこともできます。

- *startFrom* がコレクションの length 以上だった場合、-1 が返されます。これはコレクションが検索されていないことを意味します。
- *startFrom* < 0 の場合には、コレクションの終わりからのオフセットであるとみなされます (*startFrom:=startFrom+length*)。注：*startFrom* が負の値であっても、コレクションは左から右へと検索されます。
- *startFrom* = 0 の場合、コレクション全体がテストされます（デフォルト）。

例題

コレクション内で最初に合致する都市名の位置を探します：

```
var $c : Collection
var $val2;$val3 : Integer
$c:=New collection
$c.push(New object("name";"Cleveland";"zc";35049))
$c.push(New object("name";"Blountsville";"zc";35031))
$c.push(New object("name";"Adger";"zc";35006))
$c.push(New object("name";"Clanton";"zc";35046))
$c.push(New object("name";"Clanton";"zc";35045))
$val2:=$c.findIndex("FindCity";"Clanton") // $val2=3
$val3:=$c.findIndex($val2+1;"FindCity";"Clanton") // $val3=4
```

FindCity メソッドのコードは以下のとおりです：

```

var $1 : Object
var $2 : Text
$1.result:=$1.value.name=$2

```

.indexOf()

▶ 覆歴

.indexOf(*toSearch* : expression { ; *startFrom* : Integer }) : Integer

引数	タイプ		説明
<i>toSearch</i>	expression	->	コレクション内を検索する式
<i>startFrom</i>	Integer	->	検索を開始するインデックス
戻り値	Integer	<-	最初に見つかった <i>toSearch</i> のインデックス。見つからなかった場合には -1

説明

.indexof() 関数は、*toSearch* 引数の式をコレクション要素の中から検索し、最初に見つかった要素のインデックス（見つからなかった場合には -1）を返します。

このコマンドは、元のコレクションを変更しません。

toSearch パラメーターには、コレクション内で検索する式を渡します。以下のものを渡すことができます:

- スカラー値 (テキスト、数値、ブール、日付)
- null 値
- オブジェクトあるいはコレクションの参照

toSearch 引数は検出すべき要素と完全に一致している必要があります (等号演算子と同じルールが適用されます)。

オプションとして、*startFrom* 引数を渡すことで、検索を開始するコレクション要素のインデックスを指定することができます。

- startFrom* がコレクションの *length* 以上だった場合、-1 が返されます。これはコレクションが検索されていないことを意味します。
- startFrom* < 0 の場合には、コレクションの終わりからのオフセットであるとみなされます (*startFrom:=startFrom+length*)。注: *startFrom* が負の値であっても、コレクションは左から右へと検索されます。
- startFrom* = 0 の場合、コレクション全体がテストされます (デフォルト)。

例題

```

var $col : Collection
var $i : Integer
$col:=New collection(1;2;"Henry";5;3;"Albert";6;4;"Alan";5)
$i:=$col.indexOf(3) // $i=4
$i:=$col.indexOf(5;5) // $i=9
$i:=$col.indexOf("al@") // $i=5
$i:=$col.indexOf("Hello") // $i=-1

```

.indices()

▶ 覆歴

.indices(*queryString* : Text { ; ...*value* : any }) : Collection

引数	タイプ		説明
queryString	Text	->	検索条件
value	any	->	プレースホルダー使用時: 比較する値
戻り値	Collection	<-	queryString に合致するコレクション要素のインデックス

説明

`.indices()` 関数は `.query()` 関数と同様に機能しますが、*queryString* 引数の検索条件に合致する、元のコレクション要素のインデックスを返します(コレクション要素自体は返しません)。インデックスは、昇順に返されます。

このコマンドは、元のコレクションを変更しません。

queryString 引数には、以下のシンタックスを使用します:

```
propertyPath 比較演算子 値 {logicalOperator propertyPath 比較演算子 値}
```

queryString および *value* パラメーターの詳細については、[dataClass.query\(\)](#) 関数を参照ください。

例題

```
var $c; $icoll : Collection
$c:=New collection
$c.push(New object("name";"Cleveland";"zc";35049))
$c.push(New object("name";"Blountsville";"zc";35031))

$c.push(New object("name";"Adger";"zc";35006))
$c.push(New object("name";"Clanton";"zc";35046))
$c.push(New object("name";"Clanton";"zc";35045))
$icoll:=$c.indices("name = :1";"Cleveland") // $icoll=[0]
$icoll:=$c.indices("zc > 35040") // $icoll=[0,3,4]
```

.insert()

▶履歴

`.insert(index : Integer ; element : any) : Collection`

引数	タイプ		説明
index	Integer	->	要素の挿入位置
element	any	->	コレクションに挿入する要素
戻り値	Collection	<-	要素の挿入された元のコレクション

説明

`.insert()` 関数は、*index* で指定したコレクションインスタンスの位置に *element* 要素を挿入し、変更された元のコレクションを返します。

このコマンドは、元のコレクションを変更します。

index パラメーターには、コレクション内で要素を挿入する位置を渡します。

警告: コレクション要素は 0 起点である点に注意してください。

- 指定した *index* がコレクションの *length* より大きい場合、実際の開始インデックスはコレクションの *length* に設定されます。

- *index < 0* の場合、*index:=index+length* として再計算されます（コレクションの終端からのオフセットであるとみなされます）。
- 計算結果も負の値である場合、*index* は 0 に設定されます。

コレクションが受け入れるものであれば、どんな型の要素も（たとえば他のコレクションでも）挿入可能です。

例題

```
var $col : Collection
$col:=New collection("a";"b";"c";"d") // $col=["a","b","c","d"]
$col.insert(2;"X") // $col=["a","b","X","c","d"]
$col.insert(-2;"Y") // $col=["a","b","X","Y","c","d"]
$col.insert(-10;"Hi") // $col=["Hi","a","b","X","Y","c","d"]
```

.join()

▶履歴

.join(delimiter : Text { ; option : Integer }) : Text

引数	タイプ		説明
delimiter	Text	->	要素間に用いる区切り文字
option	Integer	->	<code>ck ignore null or empty</code> : 戻り値に null と空の文字列を含めない
戻り値	Text	<-	区切り文字を使ってコレクションの全要素をつなげた文字列

説明

.join() 関数は、*delimiter* に渡した文字列を区切り文字として、コレクションの全要素を一つの文字列につなげます。戻り値はつなげられた文字列です。

このコマンドは、元のコレクションを変更しません。

デフォルトで、コレクションの null あるいは空の要素も戻り値の文字列に含めます。これらを戻り値の文字列に含めたくない場合は、*option* パラメーターに `ck ignore null or empty` 定数を渡します。

例題

```
var $c : Collection
var $t1;$t2 : Text
$c:=New collection(1;2;3;"Paris";Null;"";4;5)
$t1:=$c.join("|") // 1|2|3|Paris|null||4|5
$t2:=$c.join("|";ck ignore null or empty) // 1|2|3|Paris|4|5
```

.lastIndexOf()

▶履歴

.lastIndexOf(toSearch : expression { ; startFrom : Integer }) : Integer

引数	タイプ		説明
toSearch	expression	->	コレクション内を検索する要素
startFrom	Integer	->	検索を開始するインデックス
戻り値	Integer	<-	最後に見つかった <i>toSearch</i> のインデックス。見つからなかった場合には -1

説明

`.lastIndex0f()` 関数は、`toSearch` 引数の式をコレクション要素の中から検索し、最後に見つかった要素のインデックス（見つからなかった場合には -1）を返します。

このコマンドは、元のコレクションを変更しません。

`toSearch` パラメーターには、コレクション内で検索する式を渡します。以下のものを渡すことができます：

- スカラー値 (テキスト、数値、布尔、日付)
- null 値
- オブジェクトあるいはコレクションの参照

`toSearch` 引数は検出すべき要素と完全に一致している必要があります (等号演算子と同じルールが適用されます)。

オプションとして、`startFrom` 引数を渡すことで、逆順検索を開始するコレクション要素のインデックスを指定することができます。

- `startFrom` が、コレクションの `length` から 1 を引いた数字 (`coll.length-1`) 以上の場合、コレクション全体が検索されます (デフォルト)。
- `startFrom < 0` の場合、`startFrom:=startFrom+length` として再計算されます (コレクションの終端からのオフセットであるとみなされます)。計算結果も負の値である場合、-1 が返されます。これはコレクションが検索されていないことを意味します。注: `startFrom` が負の値であっても、コレクションは右から左へと検索されます。
- `startFrom = 0` の場合、-1 が返されます。これはコレクションが検索されていないことを意味します。

例題

```
var $col : Collection
var $pos1;$pos2;$pos3;$pos4;$pos5 : Integer
$col:=Split string("a,b,c,d,e,f,g,h,i,j,e,k,e;","") // $col.length=13
npos1:=$col.lastIndex0f("e") // 戻り値: 12
npos2:=$col.lastIndex0f("e";6) // 戻り値: 4
npos3:=$col.lastIndex0f("e";15) // 戻り値: 12
npos4:=$col.lastIndex0f("e";-2) // 戻り値: 10
npos5:=$col.lastIndex0f("x") // 戻り値: -1
```

.length

▶ 履歴

`.length` : Integer

説明

`.length` プロパティは、コレクション内の要素数を返します。

`.length` プロパティは、コレクション作成時に初期化されます。要素を追加・削除すると、必要に応じて `length` は更新されます。このプロパティは読み取り専用 です (これを使用してコレクションのサイズを設定することはできません)。

例題

```
var $col : Collection // $col.length が 0 に初期化されます
$col:=New collection("one";"two";"three") // $col.length が 3 に更新されます
$col[4]:="five" // $col.length が 5 に更新されます
$vSize:=$col.remove(0;3).length // $vSize=2
```

.map()

▶ 履歴

`.map(methodName : Text { ; ...param : any }) : Collection`

引数	タイプ		説明
methodName	Text	->	コレクション要素を変換するのに使用するメソッド名
param	any	->	methodName に渡す引数
戻り値	Collection	<-	変換された値を格納する新しいコレクション

説明

`.map()` 関数は、元のコレクションの各要素に対して `methodName` メソッドを呼び出した結果に基づいた、新しいコレクションを作成します。オプションで、`param` パラメーターに、`methodName` に渡す引数を指定することができます。`.map()` は常に、元のコレクションと同じサイズのコレクションを返します。

このコマンドは、元のコレクションを変更しません。

`methodName` には、コレクション要素の評価に使用するメソッド名を渡します。`param` には、必要に応じて引数を渡します（任意）。`methodName` で指定したメソッドはどんな処理でも実行でき、引数はあってもなくても構いません。

`methodName` で指定したメソッドは以下の引数を受け取ります：

- `$1.value` (任意の型): マップする要素の値
- `in $2` (任意の型): `param`
- `in $N...` (任意の型): `paramN...`

`methodName` で指定したメソッドでは、以下の引数を設定します：

- `$1.result` (任意の型): 結果のコレクションに追加する、変換された値
- `$1.stop` (ブール): メソッドコールバックを止める場合には `true`。返された値は最後に計算されたものです。

例題

```
var $c; $c2 : Collection
$c:=New collection(1;4;9;10;20)
$c2:=$c.map("Percentage";$c.sum())
//c2=[2.27,9.09,20.45,22.73,45.45]
```

`Percentage` メソッドのコードは以下のとおりです：

```
var $1 : Object
var $2 : Real
$1.result:=Round(( $1.value/$2)*100;2)
```

.max()

▶ 履歴

`.max({ propertyPath : Text }) : any`

引数	タイプ		説明
propertyPath	Text	->	評価するオブジェクトプロパティのパス
戻り値	Boolean, Text, Number, Collection, Object, Date	<-	コレクション内の最大値

説明

`.max()` 関数は、コレクション内の最大値を持つ要素を返します（`.sort()` 関数を使用して昇順に並べ替えたときのコレクションの最後の要素が最大値の要素です）。

このコマンドは、元のコレクションを変更しません。

コレクションが異なる型の値を格納している場合、`.max()` 関数は型のリスト順の、最後の型の最大値を返します（[.sort\(\)](#) 参照）。

コレクションがオブジェクトを格納している場合には、最大値を取得するオブジェクトプロパティのパスを `propertyPath` に渡します。

コレクションが空の場合、`.max()` は *Undefined* を返します。

例題

```
var $col : Collection
$col:=New collection(200;150;55)
$col.push(New object("name";"Smith";"salary";10000))
$col.push(New object("name";"Wesson";"salary";50000))
$col.push(New object("name";"Alabama";"salary";10500))
$max:=$col.max() //{name:Alabama,salary:10500}
$maxSal:=$col.max("salary") //50000
$maxName:=$col.max("name") //"Wesson"
```

.min()

▶ 履歴

`.min({ propertyPath : Text }) : any`

引数	タイプ		説明
propertyPath	テキスト	->	評価するオブジェクトプロパティのパス
戻り値	Boolean, Text, Number, Collection, Object, Date	<-	コレクション内の最小値

説明

`.min()` 関数は、コレクション内の最小値を持つ要素を返します（[.sort\(\)](#) 関数を使用して昇順に並べ替えたときのコレクションの先頭の要素が最小値の要素です）。

このコマンドは、元のコレクションを変更しません。

コレクションが異なる型の値を格納している場合、`.min()` 関数は型のリスト順の、最初の型の最小値を返します（[.sort\(\)](#) 参照）。

コレクションがオブジェクトを格納している場合には、最小値を取得するオブジェクトプロパティのパスを `propertyPath` に渡します。

コレクションが空の場合、`.min()` は *Undefined* を返します。

例題

```
var $col : Collection
$col:=New collection(200;150;55)
$col.push(New object("name";"Smith";"salary";10000))
$col.push(New object("name";"Wesson";"salary";50000))
$col.push(New object("name";"Alabama";"salary";10500))
$min:=$col.min() //55
$minSal:=$col.min("salary") //10000
$minName:=$col.min("name") //"Alabama"
```

.orderBy()

▶ 履歴

```
.orderBy( ) : Collection
.orderBy( pathStrings : Text ) : Collection
.orderBy( pathObjects : Collection ) : Collection
.orderBy( ascOrDesc : Integer ) : Collection
```

引数	タイプ		説明
pathStrings	テキスト	->	コレクションの並べ替え基準とするプロパティパス
pathObjects	コレクション	->	条件オブジェクトのコレクション
ascOrDesc	整数	->	<code>ck ascending</code> または <code>ck descending</code> (スカラー値)
戻り値	コレクション	<-	並べ替えられたコレクションのコピー (シャロウ・コピー)

説明

`.orderBy()` 関数は、コレクションの要素を指定順に並べ替えた新しいコレクションを返します。

この関数は シャロウ・コピー を返します。つまり、元のコレクションにオブジェクト要素やコレクション要素が含まれていた場合、それらの参照は戻り値のコレクションで共有されます。また、元のコレクションが共有コレクションであった場合、返されるコレクションもまた共有コレクションになります。

このコマンドは、元のコレクションを変更しません。

引数を渡さなかった場合、メソッドはコレクション内のスカラー値を昇順に並べ替えます (オブジェクトやコレクションなどの他の型は並べ替えされないまま返されます)。この自動並べ替え順は、`ascOrDesc` パラメーターに `ck ascending` あるいは `ck descending` 定数を渡すことで変更できます (以下参照)。

また、引数を渡すことで、コレクション要素をどのように並べ替えるかを指定することもできます。次の 3つのシンタックスがサポートされています:

- `pathStrings` : Text (フォーミュラ)。シンタックス: `propertyPath1 {desc または asc}, propertyPath2 {desc または asc}, ...` (デフォルトの並び順: asc)。`pathStrings` はカンマで区切られた、1~n のプロパティパスと並び順 (任意) で構成されたフォーミュラを格納します。プロパティを渡す順番が、コレクション要素の並べ替えの優先順位を決定します。デフォルトでは、プロパティは昇順に並べ替えられます。並び順を設定するには、プロパティパスの後に半角スペースで区切ったあとに、昇順を指定するには "asc"、降順を指定するには "desc" を渡します。
- `pathObjects` : Collection。`pathObjects` コレクションには必要な数だけオブジェクトを追加することができます。デフォルトでは、プロパティは昇順に並べ替えられます ("descending" は false)。コレクションの各要素は、以下の構造を持つオブジェクトを格納します:

```
{
  "propertyPath": string,
  "descending": boolean
}
```

- `ascOrDesc` : Integer。Objects and collections テーマから、以下の定数のいずれか一つを渡します:

定数	タイプ	値	説明
<code>ck ascending</code>	Longint	0	要素は昇順に並べられます (デフォルト)
<code>ck descending</code>	Longint	1	要素は降順に並べられます

このシンタックスは、コレクション内のスカラー値のみを並べ替えます (オブジェクトやコレクションなどの他の型は並べ替えされないまま返されます)。

コレクションが異なる型の要素を格納している場合、それらはまず型ごとにグループ分けされ、そのあとで並べ替えられます。型は以下の順番で返されます:

1. null
2. ブール
3. 文字列
4. 数値
5. オブジェクト
6. コレクション

7. 日付

例題 1

数値のコレクションを昇順および降順に並べ替えます:

```
var $c; $c2; $3 : Collection
$c:=New collection
For($vCounter;1;10)
    $c.push(Random)
End for
$c2:=$c.orderBy(ck ascending)
$c3:=$c.orderBy(ck descending)
```

例題 2

オブジェクトのコレクションを、テキストフォーミュラに指定したプロパティ名に基づいて並べ替えます:

```
var $c; $c2 : Collection
$c:=New collection
For($vCounter;1;10)
    $c.push(New object("id";$vCounter;"value";Random))
End for
$c2:=$c.orderBy("value desc")
$c2:=$c.orderBy("value desc, id")
$c2:=$c.orderBy("value desc, id asc")
```

オブジェクトのコレクションをプロパティパスで並べ替えます:

```
var $c; $c2 : Collection
$c:=New collection
$c.push(New object("name";"Cleveland";"phones";New object("p1";"01";"p2";"02")))
$c.push(New object("name";"Blountsville";"phones";New object("p1";"00";"p2";"03")))
$c2:=$c.orderBy("phones.p1 asc")
```

例題 3

オブジェクトのコレクションを、*pathObjects* コレクションを使用して並べ替えます:

```
var $crit; $c; $c2 : Collection
$crit:=New collection
$c:=New collection
For($vCounter;1;10)
    $c.push(New object("id";$vCounter;"value";Random))
End for
$crit.push(New object("propertyPath";"value";"descending";True))
$crit.push(New object("propertyPath";"id";"descending";False))
$c2:=$c.orderBy($crit)
```

プロパティパスで並べ替えます:

```

var $crit; $c; $c2 : Collection
$c:=New collection
$c.push(New object("name";"Cleveland";"phones";New object("p1";"01";"p2";"02")))
$c.push(New object("name";"Blountsville";"phones";New object("p1";"00";"p2";"03")))
$crit:=New collection(New object("propertyPath";"phones.p2";"descending";True))
$c2:=$c.orderBy($crit)

```

.orderByMethod()

▶ 覆歴

.orderByMethod(*methodName* : Text { ; ...*extraParam* : expression }) : Collection

引数	タイプ		説明
<i>methodName</i>	テキスト	->	並べ替え順の指定に使用するメソッド名
<i>extraParam</i>	式	->	<i>methodName</i> に渡す引数
戻り値	コレクション	<-	並べ替えられたコレクションのコピー（シャロウ・コピー）

説明

.orderByMethod() 関数は、*methodName* メソッドを通して定義された順番でコレクション要素を並べ替えた新しいコレクションを返します。

この関数は シャロウ・コピー を返します。つまり、元のコレクションにオブジェクト要素やコレクション要素が含まれていた場合、それらの参照は戻り値のコレクションで共有されます。また、元のコレクションが共有コレクションであった場合、返されるコレクションもまた共有コレクションになります。

このコマンドは、元のコレクションを変更しません。

methodName には、二つの値を比較して、最初の値が二つ目の値より低い場合に *\$1.result* に true を返す比較メソッドの名称を渡します。必要であれば *methodName* に追加の引数を渡すこともできます。

- *methodName* で指定したメソッドは以下の引数を受け取ります:
 - \$1 (オブジェクト):
 - *\$1.value* (任意の型): 比較する一つ目の要素の値
 - *\$1.value2* (任意の型): 比較する二つ目の要素の値
 - \$2...\$N (任意の型): 追加の引数
- *methodName* で指定したメソッドでは、以下の引数を設定します:
 - *\$1.result* (ブール): *\$1.value < \$1.value2* の場合は true、それ以外は false

例題 1

文字列のコレクションをアルファベット順ではなく、数値順に並べ替えます:

```

var $c; $c2; $c3 : Collection
$c:=New collection
$c.push("33";"4";"1111";"222")
$c2:=$c.orderBy() // $c2=[ "1111", "222", "33", "4" ], アルファベット順
$c3:=$c.orderByMethod("NumAscending") // $c3=[ "4", "33", "222", "1111" ]

```

NumAscending メソッドのコードは以下のとおりです:

```

$c1.result:=Num($1.value)<Num($1.value2)

```

例題 2

文字列のコレクションを、文字列の長さを基準に並べ替えます:

```

var $fruits; $c2 : Collection
$fruits:=New collection("Orange";"Apple";"Grape";"pear";"Banana";"fig";"Blackberry";"Passion fruit")
$c2:=$fruits.orderByMethod("WordLength")
// $c2=[Passion fruit,Blackberry,Orange,Banana,Apple,Grape,pear,fig]

```

WordLength メソッドのコードは以下のとおりです:

```

$1.result:=Length(String($1.value))>Length(String($1.value2))

```

例題 3

文字コード順またはアルファベット順にコレクション要素を並べ替えます:

```

var $strings1; $strings2 : Collection
$strings1:=New collection("Alpha";"Charlie";"alpha";"bravo";"Bravo";"charlie")

// 文字コード順:
$strings2:=$strings1.orderByMethod("sortCollection";sk character codes)
// 結果 : ["Alpha", "Bravo", "Charlie", "alpha", "bravo", "charlie"]

// アルファベット順:
$strings2:=$strings1.orderByMethod("sortCollection";sk strict)
// 結果 : ["alpha", "Alpha", "bravo", "Bravo", "charlie", "Charlie"]

```

sortCollection メソッドのコードは以下のとおりです:

```

var$1Object
var$2Integer // 並べ替えオプション

$1.result:=(Compare strings($1.value;$1.value2;$2)<0)

```

.pop()

▶履歴

.pop() : any

引数	タイプ		説明
戻り値	any	<-	コレクションの最後の要素

説明

.pop() 関数は、コレクションから最後の要素を取り除き、それを戻り値として返します。

このコマンドは、元のコレクションを変更します。

空のコレクションに適用した場合、.pop() は *Undefined を返します。

例題

.pop() を .push() と組み合わせて使用すると、スタック（後入れ先出し構造）を実装することができます:

```

var $stack : Collection
$stack:=New collection //$/stack=[]
$stack.push(1;2) //$/stack=[1,2]
$stack.pop() //$/stack=[1] 、戻り値は 2 です
$stack.push(New collection(4;5)) //$/stack=[[1,[4,5]]
$stack.pop() //$/stack=[1] 、戻り値は [4,5] です
$stack.pop() //$/stack=[] 、戻り値は 1 です

```

.push()

▶ 履歴

.push(*element* : any { ;...*elementN* }) : Collection

引数	タイプ		説明
<i>element</i>	Mixed	->	コレクションに追加する要素
戻り値	コレクション	<-	要素の追加された元のコレクション

説明

.push() 関数は、一つ以上の *element* 引数をコレクションインスタンスの最後に追加し、変更された元のコレクションを返します。

このコマンドは、元のコレクションを変更します。

例題 1

```

var $col : Collection
$col:=New collection(1;2) //$/col=[1,2]
$col.push(3) //$/col=[1,2,3]
$col.push(6;New object("firstname";"John";"lastname";"Smith"))
//$/col=[1,2,3,6,{firstname:John,lastname:Smith}]

```

例題 2

戻り値のコレクションを並び替えます:

```

var $col; $sortedCol : Collection
$col:=New collection(5;3;9) //$/col=[5,3,9]
$sortedCol:=$col.push(7;50).sort()
//$/col=[5,3,9,7,50]
//$/sortedCol=[3,5,7,9,50]

```

.query()

▶ 履歴

.query(*queryString* : Text ; ...*value* : any) : Collection
 .query(*queryString* : Text ; *querySettings* : Object) : Collection

引数	タイプ		説明
queryString	テキスト	->	検索条件
value	Mixed	->	プレースホルダー使用時: 比較する値
querySettings	オブジェクト	->	クエリオプション: parameters, attributes 他
戻り値	コレクション	<-	queryString に合致するコレクション要素

説明

.query() 関数は、*queryString* および、任意の *value* や *querySettings* パラメーターによって定義された 検索条件に合致するオブジェクトコレクションの要素をすべて返します。また、元のコレクションが共有コレクションであった場合、返されるコレクションもまた共有コレクションになります。

このコマンドは、元のコレクションを変更しません。

queryString 引数には、以下のシンタックスを使用します:

```
propertyPath 比較演算子 値 {logicalOperator propertyPath 比較演算子 値}
```

queryString および *value* や *querySettings* パラメーターを使ってクエリをビルドする方法の詳細については、[DataClass.query\(\)](#) 関数を参考ください。

queryString 引数および *formula* オブジェクト引数の使用に関わらず、フォーミュラは `collection.query()` 関数でサポートされません。

例題 1

```
var $c; $c2; $c3 : Collection
$c:=New collection
$c.push(New object("name";"Cleveland";"zc";35049))
$c.push(New object("name";"Blountsville";"zc";35031))
$c.push(New object("name";"Adger";"zc";35006))
$c.push(New object("name";"Clanton";"zc";35046))
$c.push(New object("name";"Clanton";"zc";35045))
$c2:=$c.query("name = :1";"Cleveland") // $c2=[{name:Cleveland,zc:35049}]
$c3:=$c.query("zc > 35040") // $c3=[{name:Cleveland,zc:35049},{name:Clanton,zc:35046},{name:Clanton,zc:35045}]
```

例題 2

```
var $c : Collection
$c:=New collection
$c.push(New object("name";"Smith";"dateHired";!22-05-2002!;"age";45))
$c.push(New object("name";"Wesson";"dateHired";!30-11-2017!))
$c.push(New object("name";"Winch";"dateHired";!16-05-2018!;"age";36))

$c.push(New object("name";"Sterling";"dateHired";!10-5-1999!;"age";Null))
$c.push(New object("name";"Mark";"dateHired";!01-01-2002!))
```

上記のオブジェクトに対し、以下のクエリは名前に "in" が含まれている人物を返します:

```
$col:=$c.query("name = :1";"@in@")
// $col=[{name:Winch...},{name:Sterling...}]
```

以下のクエリは、変数に格納した文字列（ユーザーが入力した文字列など）から名前が始まらない人物を返します：

```
$col:=$c.query("name # :1;$aString+"@")
//if $astrig="W"
//$col=[{name:Smith...},{name:Sterling...},{name:Mark...}]
```

以下のクエリは、年齢が不明な（プロパティが null あるいは undefined に設定されている）人物を返します：

```
$col:=$c.query("age=null") // "null" ではプレースホルダーは使えません
//$col=[{name:Wesson...},{name:Sterling...},{name:Mark...}]
```

以下のクエリは、採用から90日を超える人物を返します：

```
$col:=$c.query("dateHired < :1;(Current date-90)")
//$col=[{name:Smith...},{name:Sterling...},{name:Mark...}] (今日が 01/10/2018 の場合)
```

例題 3

追加のクエリ例については、[dataClass.query\(\)](#) を参照してください。

.reduce()

▶履歴

.reduce(*methodName* : Text) : any

.reduce(*methodName* : Text ; *initValue* : any { ; ...*param* : expression }) : any

引数	タイプ		説明
<i>methodName</i>	テキスト	->	コレクション要素を処理するのに使用するメソッド名
<i>initValue</i>	Text, Number, Object, Collection, Date, Boolean	->	<i>methodName</i> の最初の呼び出しに最初の引数として使用する値
<i>param</i>	式	->	<i>methodName</i> に渡す引数
戻り値	Text, Number, Object, Collection, Date, Boolean	<-	アキュムレーター値の結果

説明

`.reduce()` 関数は、*methodName* コールバックメソッドをアキュムレーターおよびコレクションの各要素に（左から右へ）適用して、単一の値にまとめます。

このコマンドは、元のコレクションを変更しません。

methodName には、コレクション要素の評価に使用するメソッド名を渡します。*param* には、必要に応じて引数を渡します（任意）。*methodName* はコレクションの各要素を受け取り、任意の処理を実行して、結果を `$1.accumulator` に蓄積します。この値は最終的に `$1.value` に返されます。

initValue に引数を渡すことで、アキュムレーターを初期化することができます。省略された場合は、`$1.accumulator` は *Undefined* から開始されます。

methodName で指定したメソッドは以下の引数を受け取ります：

- `$1.value`: 処理する要素の値
- `in $2: param`
- `in $N...: paramN...`

methodName で指定したメソッドでは、以下の引数を設定します：

- `$1.accumulator`: メソッドで変更する値。`initValue` によって初期化します。
- `$1.stop` (ブール、任意): メソッドコールバックを止める場合には `true`。返された値は最後に計算されたものです。

例題 1

```
C_COLLECTION($c)
$c:=New collection(5;3;5;1;3;4;4;6;2;2)
$r:=$c.reduce("Multiply";1) // 戻り値は 86400 です
```

Multiply メソッドのコードは以下のとおりです:

```
If(Value type($1.value)=Is real)
    $1.accumulator:=$1.accumulator*$1.value
End if
```

例題 2

複数のコレクション要素を単一の値にまとめます:

```
var $c;$r : Collection
$c:=New collection
$c.push(New collection(0;1))
$c.push(New collection(2;3))
$c.push(New collection(4;5))
$c.push(New collection(6;7))
$r:=$c.reduce("Flatten") // $r=[0,1,2,3,4,5,6,7]
```

Flatten メソッドのコードは以下のとおりです:

```
If($1.accumulator=Null)
    $1.accumulator:=New collection
End if
$1.accumulator.combine($1.value)
```

.remove()

▶履歴

`.remove(index : Integer { ; howMany : Integer }) : Collection`

引数	タイプ		説明
index	整数	->	削除を開始する要素の位置
howMany	整数	->	削除する要素の数、省略時は 1要素を削除
戻り値	コレクション	<-	要素が削除された元のコレクション

説明

`.remove()` 関数は、`index` で指定した位置から一つまたは複数のコレクション要素を削除し、変更されたコレクションを返します。

このコマンドは、元のコレクションを変更します。

`index` パラメーターには、削除するコレクション要素の位置を渡します。

警告: コレクション要素は 0 起点である点に注意してください。指定した `index` がコレクションの `length` より大きい場合、実際の開始イン

デックスはコレクションの `length` に設定されます。

- `index < 0` の場合、`index:=index+length` として再計算されます（コレクションの終端からのオフセットであるとみなされます）。
- 計算結果も負の値である場合、`index` は 0 に設定されます。
- 計算結果がコレクションの `length` より大きい場合には、`index` は `length` に設定されます。

`howMany` には、`index` の位置から削除する要素の数を渡します。`howMany` が省略された場合、1つの要素のみが削除されます。

空のコレクションから要素を削除しようとした場合、関数は何もしません（エラーは生成されません）。

例題

```
var $col : Collection
$col:=New collection("a";"b";"c";"d";"e";"f";"g";"h")
$col.remove(3) // $col=["a","b","c","e","f","g","h"]
$col.remove(3;2) // $col=["a","b","c","g","h"]
$col.remove(-8;1) // $col=["b","c","g","h"]
$col.remove(-3;1) // $col=["b","g","h"]
```

.resize()

▶ 履歴

`.resize(size : Integer { ; defaultValue : any }) : Collection`

引数	タイプ		説明
size	整数	->	コレクションの新しいサイズ
defaultValue	Number, Text, Object, Collection, Date, Boolean	->	新規要素のデフォルト値
戻り値	コレクション	<-	リサイズされた元のコレクション

説明

`.resize()` 関数は、コレクションの `length` を引数で指定されたサイズに設定し、変更された元のコレクションを返します。

このコマンドは、元のコレクションを変更します。

- `size < length` の場合、余分な要素はコレクションから削除されます。
- `size > length` の場合、不足分の要素がコレクションに追加されます。

デフォルトで、新規要素には `null` 値が格納されます。`defaultValue` に引数を渡すことで、新規要素の値を指定することができます。

例題

```
var $c : Collection
$c:=New collection
$c.resize(10) // $c=[null,null,null,null,null,null,null,null,null,null]

$c:=New collection
$c.resize(10;0) // $c=[0,0,0,0,0,0,0,0,0,0]

$c:=New collection(1;2;3;4;5)
$c.resize(10;New object("name";"X")) // $c=[1,2,3,4,5,{name:X},{name:X},{name:X},{name:X},{name:X}]

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

.reverse()

▶履歴

.reverse() : Collection

引数	タイプ		説明
戻り値	コレクション	<-	逆順に要素を格納した新しいコレクション

説明

.reverse() 関数は、全要素が逆順になった、コレクションのディープ・コピーを返します。また、元のコレクションが共有コレクションであった場合、返されるコレクションもまた共有コレクションになります。

このコマンドは、元のコレクションを変更しません。

例題

```
var $c; $c2 : Collection  
$c:=New collection(1;3;5;2;4;6)  
$c2:=$c.reverse() // $c2=[6,4,2,5,3,1]
```

.shift()

▶履歴

.shift() : any

引数	タイプ		説明
戻り値	any	<-	コレクションの先頭要素

説明

.shift() 関数は、コレクションの先頭要素を取り除き、それを戻り値として返します。

このコマンドは、元のコレクションを変更します。

コレクションが空の場合、関数はなにもしません。

例題

```
var $c : Collection  
var $val : Variant  
$c:=New collection(1;2;4;5;6;7;8)  
$val:=$c.shift()  
// $val=1  
// $c=[2,4,5,6,7,8]
```

.slice()

▶履歴

.slice(*startFrom* : Integer { ; *end* : Integer }) : Collection

引数	タイプ		説明
startFrom	整数	->	開始インデックス (含まれる)
end	整数	->	終了インデックス (含まれない)
戻り値	コレクション	<-	抜粋要素を格納した新しいコレクション(シャロウ・コピー)

説明

.slice() 関数は、*startFrom* の位置 (含まれる) から *end* の位置 (含まれない) までのコレクションの一部を、新しいコレクションの中に返します。この関数は シャロウ・コピー を返します。また、元のコレクションが共有コレクションであった場合、返されるコレクションもまた共有コレクションになります。

このコマンドは、元のコレクションを変更しません。

戻り値のコレクションには、*startFrom* 引数で指定した要素 (含まれる) から、*end* 引数で指定した要素まで (含まれない) の全要素が格納されます。*startFrom* 引数のみを渡した場合には、*startFrom* 引数で指定した要素から最後の要素までが戻り値のコレクションに格納されます。

- *startFrom < 0* の場合、*startFrom:=startFrom+length* として再計算されます (コレクションの終端からのオフセットであるとみなされます)。
- 再計算された値も負の値だった場合、*startFrom* は 0 に設定されます。
- *end < 0* の場合、それは *end:=end+length* として再計算されます。
- 渡された値、あるいは再計算された値が *end < startFrom* の場合、関数はなにもしません。

例題

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

.some()

▶履歴

.some(*methodName* : Text { ; ...*param* : any }) : Boolean
 .some(*startFrom* : Integer ; *methodName* : Text { ; ...*param* : any }) : Boolean

引数	タイプ		説明
startFrom	整数	->	テストを開始するインデックス
methodName	テキスト	->	テストに呼び出すメソッド名
param	Mixed	->	<i>methodName</i> に渡す引数
戻り値	ブール	<-	少なくとも一つの要素がテストをパスすれば true

説明

.some() 関数は、少なくとも一つのコレクション要素が、*methodName* に指定したメソッドで実装されたテストにパスした場合に true を返します。

methodName には、コレクション要素の評価に使用するメソッド名を渡します。*param* には、必要に応じて引数を渡します (任意)。*methodName* で指定したメソッドはどんなテストでも実行でき、引数はあってなくとも構いません。このメソッドは \$1 にオブジェクトを受け取り、テストをパスした最初の要素の \$1.result を true に設定しなければなりません。

methodName で指定したメソッドは以下の引数を受け取ります:

- \$1.value: 評価する要素の値

- \$2: param
- \$N...: param2...paramN

methodName で指定したメソッドでは、以下の引数を設定します：

- \$1.result (ブール)：要素の値の評価が成功した場合には true、それ以外は false
- \$1.stop (ブール、任意)：メソッドコールバックを止める場合には true。返された値は最後に計算されたものです。

.some() 関数は、\$1.result に true を返す最初のコレクション要素を見つけると、*methodName* メソッドの呼び出しをやめて true を返します。

デフォルトでは、.some() はコレクション全体をテストします。オプションとして、startFrom 引数を渡すことで、テストを開始するコレクション要素のインデックスを指定することができます。

- startFrom がコレクションの length 以上だった場合、false が返されます。これはコレクションがテストされていないことを意味します。
- startFrom < 0 の場合には、コレクションの終わりからのオフセットであるとみなされます。
- startFrom = 0 の場合、コレクション全体がテストされます（デフォルト）。

例題

```
var $c : Collection
var $b : Boolean
$c:=New collection
$c.push(-5;-3;-1;-4;-6;-2)
$b:=$c.some("NumberGreaterThan0") // 戻り値は false
$c.push(1)
$b:=$c.some("NumberGreaterThan0") // 戻り値は true

$c:=New collection
$c.push(1;-5;-3;-1;-4;-6;-2)
$b:=$c.some("NumberGreaterThan0") // $b=true
$b:=$c.some(1;"NumberGreaterThan0") // $b=false
```

NumberGreaterThan0 メソッドのコードは以下のとおりです：

```
$1.result:=$1.value>0
```

.sort()

▶ 補足

.sort(*methodName* : Text { ; ...extraParam : any }) : Collection

引数	タイプ	説明
methodName	テキスト	-> 並べ替え順の指定に使用するメソッド名
extraParam	any	-> methodName に渡す引数
戻り値	コレクション	<- 並べ替えられた元のコレクション

説明

.sort() 関数は、コレクションの要素を並べ替えます。戻り値は並べ替えられた元のコレクションです。

このコマンドは、元のコレクションを変更します。

引数もなしに呼び出された場合、.sort() はスカラー値（数値、テキスト、日付、布尔）のみを並べ替えます。デフォルトでは、要素はそれぞれの型に応じて昇順で並べ替えられます。

カスタマイズされた順番や、型に関係なくコレクション要素を並べ替えたい場合には、二つの値を比較して、最初の値が二つ目の値より低い場合に \$1.result に true を返す比較メソッドの名称を *methodName* に渡します。必要であれば *methodName* に追加の引数を渡すこともできます。

- *methodName* で指定したメソッドは以下の引数を受け取ります:

- \$1 (オブジェクト):
 - *\$1.value* (任意の型): 比較する一つ目の要素の値
 - *\$1.value2* (任意の型): 比較する二つ目の要素の値
- \$2...\$N (任意の型): 追加の引数

methodName で指定したメソッドでは、以下の引数を設定します: * *\$1.result* (ブール): *\$1.value < \$1.value2* の場合は true、それ以外は false

コレクションが異なる型の要素を格納している場合、それらはまず型ごとにグループ分けされ、そのあとで並べ替えられます。型は以下の順番で返されます:

1. null
2. ブール
3. 文字列
4. 数値
5. オブジェクト
6. コレクション
7. 日付

例題 1

```
var $col; $col2 : Collection
$col:=New collection("Tom";5;"Mary";3;"Henry";1;"Jane";4;"Artie";6;"Chip";2)
$col2:=$col.sort() // $col2=[ "Artie", "Chip", "Henry", "Jane", "Mary", "Tom", 1, 2, 3, 4, 5, 6]
// $col=[ "Artie", "Chip", "Henry", "Jane", "Mary", "Tom", 1, 2, 3, 4, 5, 6]
```

例題 2

```
var $col; $col2 : Collection
$col:=New collection(10;20)
$col2:=$col.push(5;3;1;4;6;2).sort() // $col2=[1,2,3,4,5,6,10,20]
```

例題 3

```
var $col; $col2; $col3 : Collection
$col:=New collection(33;4;66;1111;222)
$col2:=$col.sort() // 数値順: [4,33,66,222,1111]
$col3:=$col.sort("numberOrder") // アルファベット順: [1111,222,33,4,66]
```

```
// numberOrder プロジェクトメソッド
var $1 : Object
$1.result:=String($1.value)<String($1.value2)
```

.sum()

▶ 履歴

.sum({ *propertyPath* : Text }) : Real

引数	タイプ		説明
<i>propertyPath</i>	テキスト	->	計算に使用するオブジェクトプロパティのパス
戻り値	実数	<-	コレクション要素の値の合計

説明

.sum() 関数は、コレクションインスタンスの全要素の値を合計して返します。

計算の対象となるのは数値のみです（他の型の要素は無視されます）。

コレクションがオブジェクトを格納している場合には、計算するオブジェクトプロパティのパスを *propertyPath* に渡します。

.sum() は以下の場合には 0 を返します：

- コレクションが空の場合
- コレクションに数値が含まれていない場合
- propertyPath* 引数で指定したパスがコレクション内で見つからない場合

例題 1

```
var $col : Collection
var $vSum : Real
$col:=New collection(10;20;"Monday";True;2)
$vSum:=$col.sum() //3
```

例題 2

```
var $col : Collection
var $vSum : Real
$col:=New collection
$col.push(New object("name";"Smith";"salary";10000))
$col.push(New object("name";"Wesson";"salary";50000))
$col.push(New object("name";"Gross";"salary";10500,5))
$vSum:=$col.sum("salary") //vSum=70500,5
```

.unshift()

▶履歴

.unshift(*value* : any { ;...*valueN* : any }) : Collection

引数	タイプ		説明
<i>value</i>	Text, Number, Object, Collection, Date	->	コレクションの先頭に挿入する値
戻り値	実数	<-	要素の追加された元のコレクション

説明

.unshift() 関数は、一つ以上の *value* 引数をコレクションインスタンスの先頭に挿入し、変更された元のコレクションを返します。

このコマンドは、元のコレクションを変更します。

複数の値が渡された場合、それらは一度に挿入されます。つまり、引数の順番と同じ順番で変更後のコレクションに格納されます。

例題

```
var $c : Collection
$c:=New collection(1;2)
$c.unshift(4) // $c=[4,1,2]
$c.unshift(5) // $c=[5,4,1,2]
$c.unshift(6;7) // $c=[6,7,5,4,1,2]
```


CryptoKey

4D ランゲージの `CryptoKey` クラスは、非対称の暗号化キーペアをカプセル化します。

このクラスは `4D` クラストアより提供されます。

例題

たとえば ES256 JSON Web Token (JWT) を作成するために新規 ECDSA キーペアを使ってメッセージの署名と検証をおこないます。

```
// 新規 ECDSA キーペアの生成
$key:=4D.CryptoKey.new(New object("type";"ECDSA";"curve";"prime256v1"))

// base64 形式で署名を取得
$message:="hello world"
$signature:=$key.sign($message;New object("hash";"SHA256"))

// 署名の検証
$status:=$key.verify($message;$signature;New object("hash";"SHA256"))
ASSERT($status.success)
```

概要

<code>4D.CryptoKey.new(settings : Object) : 4D.CryptoKey</code>	暗号化キーペアをカプセル化する <code>4D.CryptoKey</code> オブジェクトを新規作成します
<code>.curve : Text</code>	キーの橙円曲線名
<code>.decrypt(message : Text ; options : Object) : Object</code>	秘密 鍵を使って <code>message</code> を復号します
<code>.encrypt(message : Text ; options : Object) : Text</code>	公開 鍵を使って <code>message</code> を暗号化します
<code>.getPrivateKey() : Text</code>	<code>CryptoKey</code> オブジェクトの秘密鍵を返します
<code>.getPublicKey() : Text</code>	<code>CryptoKey</code> オブジェクトの公開鍵を返します
<code>.sign (message : Text ; options : Text) : Text</code>	utf8 形式の <code>message</code> 文字列を署名します
<code>.size : Integer</code>	キーのサイズ (ビット単位)
<code>.type : Text</code>	キーのタイプ: "RSA", "ECDSA", "PEM"
<code>.verify(message : Text ; signature : Text ; options : Object) : object</code>	utf8 形式の <code>message</code> 文字列の署名を検証します

4D.CryptoKey.new()

▶履歴

4D.CryptoKey.new(*settings* : Object) : 4D.CryptoKey

引数	タイプ		説明
settings	Object	->	キーペアを生成・ロードするための設定
result	4D.CryptoKey	<-	暗号化キーペアをカプセル化したオブジェクト

4D.CryptoKey.new() 関数は、*settings* オブジェクト引数に基づいて暗号化キーペアをカプセル化する 4D.CryptoKey オブジェクトを新規作成します。新規の RSA または ECDSA キーを生成するほか、PEM 形式の既存のキーペアをロードすることができます。

settings

プロパティ	タイプ	説明
curve	テキスト	ECDSA 曲線名
pem	テキスト	ロードする PEM 形式の暗号化キー
size	integer	RSA キーのサイズ (ビット単位)
type	テキスト	キーのタイプ: "RSA", "ECDSA", "PEM"

CryptoKey

戻り値の CryptoKey オブジェクトは、暗号化キーペアをカプセル化します。これは共有オブジェクトのため、複数の 4D プロセスによって同時使用できます。

.curve

▶履歴

.curve : Text

ECDSA キーのみ: キーの楕円曲線名。通常、ES256 (デフォルト) の場合は "prime256v1", ES384 の場合は "secp384r1", ES512 の場合は "secp521r1"。

.decrypt()

▶履歴

.decrypt(*message* : Text ; *options* : Object) : Object

引数	タイプ		説明
message	テキスト	->	<i>options.encodingEncrypted</i> を使ってデコードし復号するメッセージ文字列
options	オブジェクト	->	デコーディングオプション
戻り値	オブジェクト	<-	ステータス

.decrypt() 関数は、秘密鍵を使って *message* を復号します。使用されるアルゴリズムはキーの種類に依存します。

キーは RSA キーでなければならず、アルゴリズムは RSA-OAEP です ([RFC 3447](#) 参照)。

options

プロパティ	タイプ	説明
hash	テキスト	使用する Digest アルゴリズム。例: "SHA256", "SHA384", "SHA512"。
encodingEncrypted	テキスト	復号するバイナリ形式に <code>message</code> を変換するためのエンコーディング。可能な値: "Base64" または "Base64URL"。デフォルト値: "Base64"
encodingDecrypted	テキスト	バイナリの復号メッセージを文字列に変換するためのエンコーディング。可能な値: "UTF-8", "Base64" または "Base64URL"。デフォルト値: "UTF-8"

戻り値

`message` の復号に成功した場合には、`success` プロパティが `true` に設定された `status` オブジェクトを返します。

プロパティ	タイプ	説明
<code>success</code>	<code>boolean</code>	メッセージの復号に成功した場合は <code>true</code>
<code>result</code>	テキスト	<code>options.encodingDecrypted</code> を使って復号およびデコードされたメッセージ
<code>errors</code>	collection	<code>success</code> が <code>false</code> の場合、エラーのコレクションが含まれている場合があります。

キーまたはアルゴリズムが合致しないなどの理由で `message` の復号に成功しなかった場合、返される `status` オブジェクトの `status.errors` プロパティにはエラーのコレクションが格納されます。

.encrypt()

▶ 履歴

`.encrypt(message : Text ; options : Object) : Text`

引数	タイプ		説明
<code>message</code>	テキスト	->	<code>options.encodingEncrypted</code> を使ってエンコードし暗号化するメッセージ文字列
<code>options</code>	オブジェクト	->	エンコーディングオプション
戻り値	テキスト	<-	<code>options.encodingEncrypted</code> を使って暗号化およびエンコードされたメッセージ

`.encrypt()` 関数は、公開 鍵を使って `message` を暗号化します。使用されるアルゴリズムはキーの種類に依存します。

キーは RSA キーでなければならず、アルゴリズムは RSA-OAEP です ([RFC 3447](#) 参照)。

options

プロパティ	タイプ	説明
<code>hash</code>	テキスト	使用する Digest アルゴリズム。例: "SHA256", "SHA384", "SHA512"。
<code>encodingEncrypted</code>	テキスト	バイナリの暗号化メッセージを文字列に変換するためのエンコーディング。可能な値: "Base64" または "Base64URL"。デフォルト値: "Base64"
<code>encodingDecrypted</code>	テキスト	暗号化するバイナリ形式に <code>message</code> を変換するためのエンコーディング。可能な値: "UTF-8", "Base64" または "Base64URL"。デフォルト値: "UTF-8"

戻り値

戻り値は暗号化されたメッセージです。

.getPrivateKey()

▶ 履歴

.getPrivateKey() : Text

引数	タイプ	説明
戻り値	テキスト	<- PEM 形式の秘密鍵

.getPrivateKey() 関数は、CryptoKey オブジェクトの秘密鍵を返します (PEM形式)。無い場合は空の文字列を返します。

戻り値

戻り値は秘密鍵です。

.getPublicKey()

▶ 履歴

.getPublicKey() : Text

引数	タイプ	説明
戻り値	テキスト	<- PEM 形式の公開鍵

.getPublicKey() 関数は、CryptoKey オブジェクトの公開鍵を返します (PEM形式)。無い場合は空の文字列を返します。

戻り値

戻り値は公開鍵です。

---# # .pem

▶ 履歴

.pem : Text

ロードする PEM 形式の暗号化キー。秘密鍵を渡した場合、RSA または ECDSA の公開鍵は秘密鍵から推定されます。

.sign()

▶ 履歴

.sign (*message* : Text ; *options* : Text) : Text

引数	タイプ	説明
message	テキスト	-> 署名をするメッセージ
options	オブジェクト	-> 署名オプション
戻り値	テキスト	<- "encoding" オプションに応じて Base64 または Base64URL 形式の署名

.sign() 関数は、CryptoKey オブジェクトキーおよび指定された *options* を使って、utf8 形式の *message* 文字列を署名します。
options.encoding 属性に指定した値に応じて、base64 または base64URL 形式の署名を返します。

CryptoKey は有効な秘密鍵を格納していなくてはなりません。

options

プロパティ	タイプ	説明
hash	テキスト	使用する Digest アルゴリズム。例: "SHA256", "SHA384", "SHA512"。JWT の生成に使われた場合、ハッシュサイズは PS@, ES@, RS@, または PS@ のアルゴリズムサイズと同じでなくてはなりません。
encodingEncrypted	テキスト	バイナリの暗号化メッセージを文字列に変換するためのエンコーディング。可能な値: "Base64" または "Base64URL"。デフォルト値: "Base64"
pss	boolean	確率的署名スキーム (PSS) を使用する。RSA キーでない場合は無視されます。PS@ アルゴリズム用の JWT を生成する場合は <code>true</code> を渡します。
encoding	テキスト	戻り値の署名のエンコード方式。可能な値: "Base64" または "Base64URL"。デフォルト値: "Base64"

戻り値

utf8 形式の `message` 文字列。

.size

▶ 履歴

`.size : Integer`

RSA キーのみ: キーのサイズ (ビット単位)。通常は 2048 (デフォルト)

.type

▶ 履歴

`.type : Text`

キーのタイプ: "RSA", "ECDSA", "PEM"

- "RSA": `settings.size` に指定されたサイズを `.size` として使った、RSA キーペア
- "ECDSA": `settings.curve` に指定された曲線を `.curve` として用いた、楕円曲線デジタル署名アルゴリズム (Elliptic Curve Digital Signature Algorithm) キーペア ECDSA キーは署名だけに使用されるもので、暗号化には使用できないことに留意してください。
- "PEM": `settings.pem` を `.pem` として使った、PEM 形式のキーペア

.verify()

▶ 履歴

`.verify(message : Text ; signature : Text ; options : Object) : object`

引数	タイプ	説明
<code>message</code>	テキスト	-> 署名生成時に使われたメッセージ文字列
<code>signature</code>	テキスト	-> 検証の対象である、 <code>options.encoding</code> に応じて Base64 または Base64URL 形式の署名
<code>options</code>	オブジェクト	-> 署名オプション
戻り値	オブジェクト	<- 検証ステータス

`.verify()` 関数は、`CryptoKey` オブジェクトキーおよび指定された `options` を使って、utf8 形式の `message` 文字列の署名を検証します。

`CryptoKey` は有効な 公開 鍵を格納していなくてはなりません。

`options`

プロパティ	タイプ	説明
hash	テキスト	使用する Digest アルゴリズム。例: "SHA256", "SHA384", "SHA512"。JWT の生成に使われた場合、ハッシュサイズは PS@, ES@, RS@, または PS@ のアルゴリズムサイズと同じでなければなりません。
pss	boolean	確率的署名スキーム (PSS) を使用する。RSA キーでない場合は無視されます。PS@ アルゴリズム用の JWT を生成する場合は true を渡します。
encoding	テキスト	署名のエンコード方式。可能な値: "Base64" または "Base64URL"。デフォルト値: "Base64"

戻り値

検証で署名が合致した場合には、`success` プロパティが `true` に設定された `status` オブジェクトを返します。

`message`、キーまたはアルゴリズムが署名と合致しないなどの理由で検証が成功しなかった場合、返される `status` オブジェクトの `status.errors` プロパティにはエラーのコレクションが格納されます。

プロパティ	タイプ	説明
<code>success</code>	boolean	署名がメッセージと合致すれば <code>true</code>
<code>errors</code>	collection	<code>success</code> が <code>false</code> の場合、エラーのコレクションが含まれている場合があります。

DataClass

データクラスはデータベーステーブルへのオブジェクトインターフェースを提供します。4Dアプリケーション内のデータクラスはすべて、ds データストア のプロパティとして利用可能です。

概要

`.attributeName : DataClassAttribute`

プロパティとして直接利用可能なオブジェクト

`.all ({ settings : Object }) : 4D.EntitySelection`

データクラスの全エンティティをエンティティセレクションとして返します

`.clearRemoteCache()`

データクラスの ORDAキャッシュを空にします

`.fromCollection(objectCol : Collection { ; settings : Object }) : 4D.EntitySelection`

`objectCol` 引数のオブジェクトのコレクションに基づいてデータクラスのエンティティを更新あるいは作成し、対応するエンティティセレクションを返します

`.get(primaryKey : Integer { ; settings : Object }) : 4D.Entity`

`.get(primaryKey : Text { ; settings : Object }) : 4D.Entity`

`primaryKey` に渡したプライマリーキーに合致するエンティティを返します

`.getCount() : Integer`

データクラスに含まれる総エンティティ数を返します

`.getDataStore() : cs.DataStore`

指定したデータクラスが属しているデータストアを返します

`. getInfo() : Object`

データクラスの情報を提供するオブジェクトを返します

`.getRemoteCache() : Object`

データクラスの ORDAキャッシュの内容を記述したオブジェクトを返します

`.new() : 4D.Entity`

メモリ内にデータクラスに空のエンティティを新規作成しそれを返します

`.newSelection({ keepOrder : Integer }) : 4D.EntitySelection`

追加可能な、空の新規エンティティセレクションをメモリ内に作成します

`.query(queryString : Text { ; ...value : any } { ; querySettings : Object }) : 4D.EntitySelection`

`.query(formula : Object { ; querySettings : Object }) : 4D.EntitySelection`

`queryString` または `formula` と任意の `value` 引数で指定した検索条件に合致するエンティティを検索します

`.setRemoteCacheSettings(settings : Object)`

データクラスの ORDAキャッシュについて、タイムアウトと最大サイズを指定します

`.attributeName`

▶履歴

`.attributeName : DataClassAttribute`

説明

データクラスの属性は、データクラスの プロパティとして直接利用可能なオブジェクト です。

戻り値は `DataClassAttribute` クラスのオブジェクトです。これらのオブジェクトが持つプロパティを読み取ることによって、データクラス属性に関する情報が取得できます。

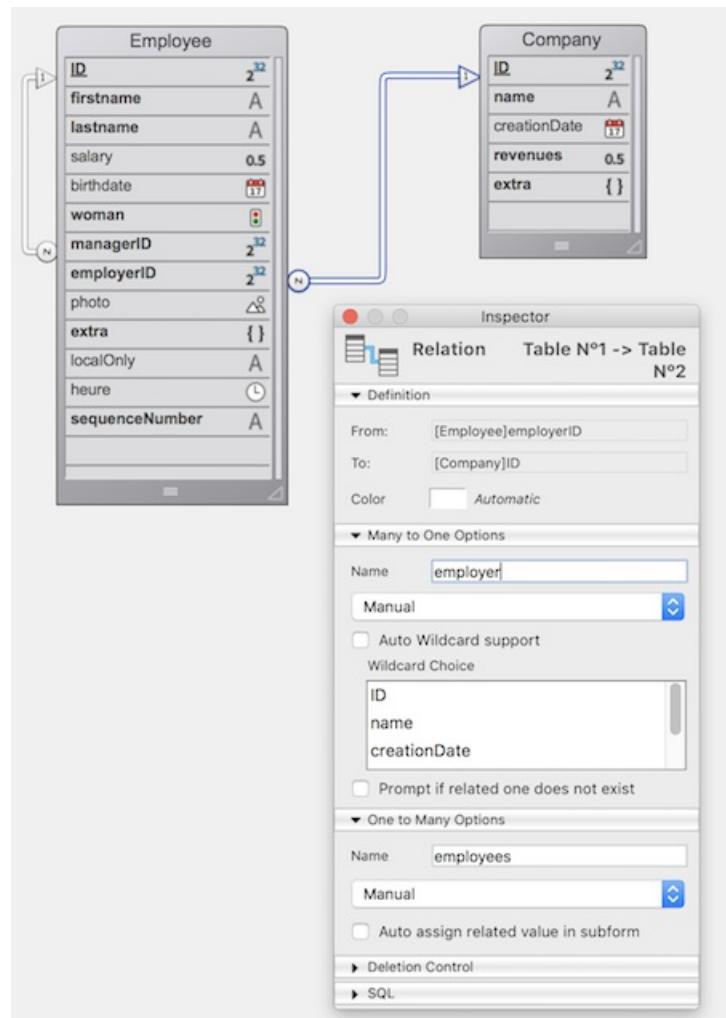
データクラス属性オブジェクトを編集することは可能ですが、元となるデータベースストラクチャーは変更されません。

例題 1

```
$salary:=ds.Employee.salary // Employeeデータクラスの salary属性を返します  
$compCity:=ds.Company["city"] // Companyデータクラスの city属性を返します
```

例題 2

以下のストラクチャーを前提とします:



```

var $firstnameAtt;$employerAtt;$employeesAtt : Object

$firstnameAtt:=ds.Employee.firstname
//{name:firstname,kind:storage,fieldType:0,type:string,fieldNumber:2,indexed:true,
//keyWordIndexed:false,autoFilled:false,mandatory:false,unique:false}

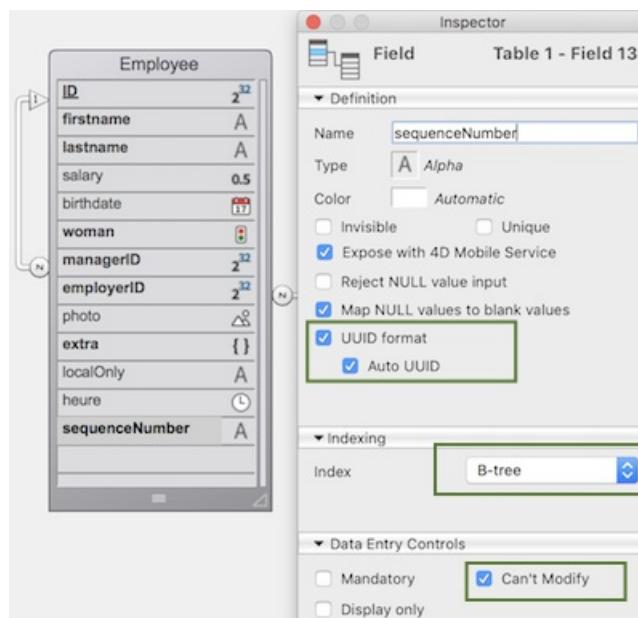
$employerAtt:=ds.Employee.employer
//{name:employer,kind:relatedEntity,relatedDataClass:Company,
//fieldType:38,type:Company,inverseName:employees}
//38=Is object

$employeesAtt:=ds.Company.employees
//{name:employees,kind:relatedEntities,relatedDataClass:Employee,
//fieldType:42,type:EmployeeSelection,inverseName:employer}
//42=Is collection

```

例題 3

以下のテーブルプロパティを前提とします:



```

var $sequenceNumberAtt : Object
$sequenceNumberAtt=ds.Employee.sequenceNumber
//{name:sequenceNumber,kind:storage,fieldType:0,type:string,fieldNumber:13,
//indexed:true,keyWordIndexed:false,autoFilled:true,mandatory:false,unique:true}

```

.all()

▶ 覆歴

.all ({ settings : Object }) : 4D.EntitySelection

引数	タイプ		説明
settings	Object	->	ビルドオプション: context
戻り値	4D.EntitySelection	<-	データクラスの全エンティティの参照

説明

.all() 関数はデータストアをクエリして、データクラスの全エンティティをエンティティセレクションとして返します。

エンティティはデフォルトの順番で返され、通常は作成順になっています。ただし、エンティティ削除後に新規追加した場合には、デフォルトの順番は作成

順を反映しない点に留意が必要です。

エンティティが見つからない場合、空のエンティティセレクションが返されます。

この関数には、レイジーローディングが適用されます。

settings

任意の *settings* パラメーターには、追加オプションを格納したオブジェクトを渡すことができます。以下のプロパティがサポートされています:

プロパティ	タイプ	説明
context	Text	エンティティセレクションに適用されている最適化コンテキストのラベル。エンティティセレクションを扱うコードはこのコンテキストを使うことで最適化の恩恵を受けます。この機能は ORDA のクライアント/サーバー処理 を想定して設計されています。

データクラス内の総エンティティ数を知るには、`ds.myClass.all().length` 式よりも最適化された `getCount()` 関数を使用することが推奨されます。

例題

```
var $allEmp : cs.EmployeeSelection  
$allEmp:=ds.Employee.all()
```

.clearRemoteCache()

▶ 覆歴

.clearRemoteCache() | 引数 | タイプ | | 説明 | | -- | --- | ::| ----- | | | | このコマンドは引数を必要としません |

説明

`.clearRemoteCache()` 関数は、データクラスの ORDAキャッシュを空にします。

この関数は `timeout` および `maxEntries` の値をリセットしません。

例題

```
var $ds : 4D.DataStoreImplementation  
var $persons : cs.PersonsSelection  
var $p : cs.PersonsEntity  
var $cache : Object  
var $info : Collection  
var $text : Text  
  
$ds:=Open datastore(New object("hostname"; "www.myserver.com"); "myDS")  
  
$persons:=$ds.Persons.all()  
$text:=""  
For each ($p; $persons)  
    $text:=$p.firstname+" lives in "+$p.address.city+" / "  
End for each  
  
$cache:=$ds.Persons.getRemoteCache()  
  
$ds.Persons.clearRemoteCache()  
// Persons データクラスのキャッシュ = {timeout:30;maxEntries:30000;stamp:255;entries:[]}
```

.fromCollection()

▶履歴

.fromCollection(*objectCol* : Collection { ; *settings* : Object }) : 4D.EntitySelection

引数	タイプ		説明
<i>objectCol</i>	コレクション	->	エンティティにマップするオブジェクトのコレクション
<i>settings</i>	オブジェクト	->	ビルドオプション: context
戻り値	4D.EntitySelection	<-	コレクションから作成したエンティティセレクション

説明

.fromCollection() 関数は、*objectCol* 引数のオブジェクトのコレクションに基づいてデータクラスのエンティティを更新あるいは作成し、対応するエンティティセレクションを返します。

objectCol パラメーターには、データクラスの既存エンティティを更新、または新規エンティティを作成するためのオブジェクトのコレクションを渡します。プロパティ名は、データクラスの属性名と同一である必要があります。プロパティ名がデータクラスに存在しない場合、それは無視されます。コレクション内で属性値が定義されていない場合、その値は null になります。

コレクションのオブジェクト要素とエンティティのマッピングは、属性名と型の合致をもって行われます。オブジェクトプロパティがエンティティ属性と同じ名前であっても、型が合致しない場合には、エンティティの属性は空のままでです。

作成モードと更新モード

objectCol 引数の各オブジェクトについて:

- オブジェクトがブール型の "__NEW" プロパティを含み、それが false に設定されている場合(あるいは "__NEW" プロパティが含まれていない場合)、オブジェクトの対応する値でエンティティが更新あるいは作成されます。プライマリーキーに関するチェックはおこなわれません:
 - プライマリーキーが指定されていて実在する場合、エンティティは更新されます。この場合、プライマリーキーはそのまま、あるいは "__KEY" プロパティを(プライマリーキー値とともに)使って指定することができます。
 - そのまま指定したプライマリーキーが実在しない場合、エンティティは作成されます。
 - プライマリーキーを指定していない場合、エンティティは作成され、標準のデータベースのルールに基づいてプライマリーキー値が割り当てられます。
- オブジェクトがブール型の "__NEW" プロパティを含み、それが true に設定されている場合、オブジェクトの対応する値でエンティティが作成されます。プライマリーキーに関するチェックがおこなわれます:
 - そのまま指定したプライマリーキーが実在する場合、エラーが返されます。
 - そのまま指定したプライマリーキーが実在しない場合、エンティティは作成されます。
 - プライマリーキーを指定していない場合、エンティティは作成され、標準のデータベースのルールに基づいてプライマリーキー値が割り当てられます。

値を格納している "__KEY" プロパティは、 "__NEW" プロパティが false に設定(あるいは省略)されていて、かつ対応するエンティティが存在する場合のみ、考慮されます。それ以外の場合には、 "__KEY" プロパティ値は無視されるため、プライマリーキーの値はそのまま渡さなければなりません。

リレートエンティティ

objectCol 引数のオブジェクトは、一つ以上のリレートエンティティに対応するオブジェクトをネストすることができます。これはエンティティ間のリンクを作成・更新するのに有用です。

リレートエンティティに相当するネストされたオブジェクトは、リレートエンティティのプライマリーキー値を格納した "__KEY" プロパティあるいはプライマリーキー属性を格納している必要があります。 "__KEY" プロパティを使用すると、プライマリーキー属性名に依存する必要がありません。

この機構によって、リレートエンティティの中身を作成・更新することはできません。

記号

"__STAMP" プロパティが指定された場合、データストアのスタンプとのチェックがおこなわれ、エラーが返されることがあります ("与えられたスタンプはテー
ブルXXX のレコード# XXのカレントのものと合致しません")。詳細については [エンティティロッキング](#) を参照ください。

settings

任意の *settings* パラメーターには、追加オプションを格納したオブジェクトを渡すことができます。以下のプロパティがサポートされています:

プロパティ	タイプ	説明
context	Text	エンティティセレクションに適用されている最適化コンテキストのラベル。エンティティセレクションを扱うコードはこのコンテキストをすることで最適化の恩恵を受けます。この機能は ORDA のクライアント/サーバー処理 を想定して設計されています。

例題 1

既存のエンティティを更新します。`__NEW` プロパティはなく、従業員のプライマリーキーは属性に実在の値を指定して渡します:

```
var $empsCollection : Collection
var $emp : Object
var $employees : cs.EmployeeSelection

$empsCollection:=New collection
$emp:=New object
$emp.ID:=668 // Employeeテーブルの実在する主キー
$emp.firstName:="Arthur"
$emp.lastName:="Martin"
$emp.employer:=New object("ID";121) // リレートデータクラス Company の実在する主キー
// リレートデータクラス Company に実在する別の主キーを指定すれば、会社を変更することができます
$empsCollection.push($emp)
$employees:=ds.Employee.fromCollection($empsCollection)
```

例題 2

既存のエンティティを更新します。`__NEW` プロパティはなく、従業員のプライマリーキーは `__KEY` プロパティに実在の値を指定して渡します:

```
var $empsCollection : Collection
var $emp : Object
var $employees : cs.EmployeeSelection

$empsCollection:=New collection
$emp:=New object
$emp.__KEY:=1720 // Employeeテーブルの実在する主キー
$emp.firstName:="John"
$emp.lastName:="Boorman"
$emp.employer:=New object("ID";121) // リレートデータクラス Company の実在する主キー
// リレートデータクラス Company に実在する別の主キーを指定すれば、会社を変更することができます
$empsCollection.push($emp)
$employees:=ds.Employee.fromCollection($empsCollection)
```

例題 3

単純に、コレクションから新しいエンティティを作成します:

```
var $empsCollection : Collection
var $emp : Object
var $employees : cs.EmployeeSelection

$empsCollection:=New collection
$emp:=New object
$emp.firstName:="Victor"
$emp.lastName:="Hugo"
$empsCollection.push($emp)
$employees:=ds.Employee.fromCollection($empsCollection)
```

例題 4

新規エンティティを作成します。`__NEW` プロパティは `true` で、従業員のプライマリーキーは指定しません：

```
var $empsCollection : Collection
var $emp : Object
var $employees : cs.EmployeeSelection

$empsCollection:=New collection
$emp:=New object
$emp.firstName:="Mary"
$emp.lastName:="Smith"
$emp.employer:=New object("__KEY";121) // リレートデータクラス Company の実在する主キー
$emp.__NEW:=True
$empsCollection.push($emp)
$employees:=ds.Employee.fromCollection($empsCollection)
```

例題 5

新規エンティティを作成します。`__NEW` プロパティはなく、従業員のプライマリーキー属性を指定しますが、その値は実在しません：

```
var $empsCollection : Collection
var $emp : Object
var $employees : cs.EmployeeSelection

$empsCollection:=New collection
$emp:=New object
$emp.ID:=10000 // 実在しない主キー
$emp.firstName:="Françoise"
$emp.lastName:="Sagan"
$empsCollection.push($emp)
$employees:=ds.Employee.fromCollection($empsCollection)
```

例題 6

2つのエンティティが同じプライマリーキーを持つ場合、最初のエンティティは作成・保存されますが、2つめのエンティティの処理は失敗します：

```

var $empsCollection : Collection
var $emp; $emp2 : Object
var $employees : cs.EmployeeSelection

$empsCollection:=New collection
$emp:=New object
$emp.ID:=10001 // 実在しない主キー
$emp.firstName:="Simone"
$emp.lastName:="Martin"
$emp._NEW:=True
$empsCollection.push($emp)

$emp2:=New object
$emp2.ID:=10001 // 上と同じ主キー
$emp2.firstName:="Marc"
$emp2.lastName:="Smith"
$emp2._NEW:=True
$empsCollection.push($emp2)
$employees:=ds.Employee.fromCollection($empsCollection)
// 最初のエンティティは作成されます
// 2つめのエンティティは重複キーでエラーになります

```

参照

[.toCollection\(\)](#)

.get()

▶履歴

.get(*primaryKey* : Integer { ; *settings* : Object }) : 4D.Entity
 .get(*primaryKey* : Text { ; *settings* : Object }) : 4D.Entity

引数	タイプ		説明
<i>primaryKey</i>	整数または文字列	->	取得するエンティティのプライマリーキー値
<i>settings</i>	オブジェクト	->	ビルドオプション: context
戻り値	4D.Entity	<-	指定したプライマリーキーに合致するエンティティ

説明

.get() 関数はデータクラスをクエリして、*primaryKey* に渡したプライマリーキーに合致するエンティティを返します。

primaryKey には、取得したいエンティティのプライマリーキーの値を渡します。値の型は、データストアで設定されたプライマリーキーの型（倍長整数あるいはテキスト）と合致している必要があります。 [.getKey\(\)](#) 関数に `dk key as string` 引数を渡すと、プライマリーキーの値が常にテキスト型で返されるように指定することができます。

primaryKey 引数のプライマリーキーを持つエンティティが見つからない場合、Null エンティティが返されます。

この関数にはレイジーローディングが適用され、リレートデータは必要な時にのみディスクから読み込まれます。

settings

任意の *settings* パラメーターには、追加オプションを格納したオブジェクトを渡すことができます。以下のプロパティがサポートされています：

プロパティ	タイプ	説明
context	Text	エンティティに適用されている自動の最適化コンテキストのラベル。エンティティを読み込む以降のコードは、このコンテキストを使うことで最適化の恩恵を受けます。この機能は ORDA のクライアント/サーバー処理 を想定して設計されています。

例題 1

```

var $entity : cs.EmployeeEntity
var $entity2 : cs.InvoiceEntity
$entity:=ds.Employee.get(167) // プライマリーキーの値が 167のエンティティを返します
$entity2:=ds.Invoice.get("DGGX20030") // プライマリーキーの値が "DGGX20030" のエンティティを返します

```

例題 2

context プロパティの使用について紹介します:

```

var $e1; $e2; $e3; $e4 : cs.EmployeeEntity
var $settings; $settings2 : Object

$settings:=New object("context";"detail")
$settings2:=New object("context";"summary")

$e1:=ds.Employee.get(1;$settings)
completeAllData($e1) // completeAllData メソッドにおいて最適化がトリガーされ、"detail" コンテキストが割り当てられます

$e2:=ds.Employee.get(2;$settings)
completeAllData($e2) // completeAllData メソッドには、"detail" コンテキストに付随する最適化が適用されます

$e3:=ds.Employee.get(3;$settings2)
completeSummary($e3) // completeSummary メソッドにおいて最適化がトリガーされ、"summary" コンテキストが割り当てられます

$e4:=ds.Employee.get(4;$settings2)
completeSummary($e4) // completeSummary メソッドには、"summary" コンテキストに付随する最適化が適用されます

```

.getCount()

▶ 覆歴

.getCount() : Integer

引数	タイプ	説明
result	整数	<- データクラスに含まれる全エンティティ数

説明

.getCount() 関数は、データクラスに含まれる総エンティティ数を返します。

トランザクション内でこの関数を使用した場合、トランザクション中に作成されたエンティティは考慮されます。

例題

```

var $ds : 4D.DataStoreImplementation
var $number : Integer

$ds:=Open datastore(New object("hostname"; "www.myserver.com"); "myDS")

$number:=$ds.Persons.getCount()

```

.getDataStore()

▶ 覆歴

.getDataStore() : cs.DataStore

引数	タイプ		説明
戻り値	cs.DataStore	<-	データクラスが属しているデータストア

説明

`.getDataStore()` 関数は、指定したデータクラスが属しているデータストアを返します。

返されるデータストアは次のいずれかです：

- `ds` コマンドによって返されるメインデータストア
- `Open datastore` コマンドを使用して開かれたリモートデータストア

例題

`SearchDuplicate` プロジェクトメソッドは、任意のデータクラス内の重複した値を検索します。

```
var $pet : cs.CatsEntity
$pet:=ds.Cats.all().first() // エンティティを取得します
SearchDuplicate($pet;"Dogs")

// SearchDuplicate メソッド
// SearchDuplicate(entity_to_search;dataclass_name)

#DECLARE ($pet : Object ; $dataClassName : Text)
var $dataStore; $duplicates : Object

$dataStore:=$pet.getDataClass().getDataStore()
$duplicates:=$dataStore[$dataClassName].query("name=:1";$pet.name)
```

.getInfo()

▶ 覆歴

`.getInfo()` : Object

引数	タイプ		説明
戻り値	オブジェクト	<-	データクラスの情報

説明

`.getInfo()` 関数は、データクラスの情報を提供するオブジェクトを返します。このメソッドは汎用的なコードを書くのに有用です。

返されるオブジェクト

プロパティ	タイプ	説明
exposed	布尔	データクラスが REST に公開されれば true
name	Text	データクラスの名称
primaryKey	テキスト	データクラスのプライマリーキー属性の名称
tableNumber	整数	内部的な 4Dテーブル番号

例題 1

```

#DECLARE ($entity : Object)
var $status : Object

computeEmployeeNumber($entity) // エンティティに対する何らかの操作

$status:=$entity.save()
if($status.success)
    ALERT("テーブル "+$entity.getDataClass().getInfo().name+" のレコードが更新されました。")
End if

```

例題 2

```

var $settings : Object
var $es : cs.ClientsSelection

$settings:=New object
$settings.parameters:=New object("receivedIds";getIds())
$settings.attributes:=New object("pk";ds.Clients.getInfo().primaryKey)
$es:=ds.Clients.query(":pk in :receivedIds";$settings)

```

例題 3

```

var $pk : Text
var $dataClassAttribute : Object

$pk:=ds.Employee.getInfo().primaryKey
$dataClassAttribute:=ds.Employee[$pk] // 必要に応じてプライマリーキー属性へのアクセスが可能です

```

参照

[DataClassAttribute.exposed](#)

.getRemoteCache()

▶ 履歴

.getRemoteCache() : Object

引数	タイプ		説明
result	オブジェクト	<-	データクラスの ORDAキャッシュの内容を記述したオブジェクト。

上級者向け: この機能は、特定の構成のため、ORDAのデフォルト機能をカスタマイズする必要がある開発者向けです。ほとんどの場合、使用する必要はないでしょう。

説明

.getRemoteCache() 関数は、データクラスの ORDAキャッシュの内容を記述したオブジェクトを返します。

4D のシングルユーザーアプリケーションからこの関数を呼び出した場合、Null が返されます。

戻り値のオブジェクトには、以下のプロパティが格納されています:

プロパティ	タイプ	説明
maxEntries	整数	エントリコレクションの最大数
stamp	整数	キャッシュのスタンプ
timeout	整数	キャッシュの新しいエントリーが期限切れとなるまでの残り時間。
entries	コレクション	キャッシュ内の各エンティティにつき、1つのエントリオブジェクトを格納します。

エントリコレクション内の各エントリオブジェクトは、以下のプロパティを持ちます：

プロパティ	タイプ	説明
data	Object	エントリーのデータを格納するオブジェクト
expired	ブール	エントリーが期限切れの場合に true
key	テキスト	エンティティのプライマリキー

各エントリーの `data` オブジェクトは、以下のプロパティを持ちます：

プロパティ	タイプ	説明
<code>__KEY</code>	String	エンティティのプライマリキー
<code>__STAMP</code>	倍長整数	データベース内のエンティティのタイムスタンプ
<code>__TIMESTAMP</code>	String	データベース内のエンティティのスタンプ (形式: YYYY-MM-DDTHH:MM:SS:ms:Z)
<code>dataClassAttributeName</code>	バリアント	データクラス属性に対応するデータがキャッシュに存在する場合、それはデータベースと同じ型のプロパティに返されます。

リレートエンティティに関するデータは、`data` オブジェクトのキャッシュに保存されます。

例題

次の例で、`$ds.Persons.all()` は、先頭エンティティをそのすべての属性とともにロードします。その後リクエストの最適化がおこなわれ、`firstname` と `address.city` のみがロードされます。

`address.city` は、`Persons` データクラスのキャッシュにロードされることに注意してください。

`Address` データクラスの先頭エンティティだけがキャッシュに格納されます。先頭エンティティは、ループの最初の繰り返しでロードされます。

```

var $ds : 4D.DataStoreImplementation
var $persons : cs.PersonsSelection
var $p : cs.PersonsEntity
var $cachePersons; $cacheAddress : Object
var $text : Text

$ds:=Open datastore(New object("hostname"; "www.myserver.com"); "myDS")

$persons:=$ds.Persons.all()

$text:=""
For each ($p; $persons)
    $text:=$p.firstname+" lives in "+$p.address.city+"
End for each

$cachePersons:=$ds.Persons.getRemoteCache()
$cacheAddress:=$ds.Adress.getRemoteCache()

```

参照

[.setRemoteCacheSettings\(\)](#)

`.clearRemoteCache()`

`.new()`

▶ 覆歴

`.new() : 4D.Entity`

引数	タイプ		説明
戻り値	4D.Entity	<-	データクラスの新規エンティティ

説明

`.new()` 関数は、メモリ内にデータクラスに空のエンティティを新規作成しそれを返します。

エンティティオブジェクトはメモリ内に作成されますが、`.save()` 関数が呼び出されるまでデータベースには保存されません。エンティティを保存する前に削除した場合、復元することはできません。

4D Server: クライアント/サーバーにおいては、対応するテーブルのプライマリーキーが自動インクリメントであった場合、エンティティがサーバー側に保存されたときに計算されます。

エンティティの全属性は `null` 値で初期化されます。

4Dデータベースのストラクチャーレベルで ヌル値を空値にマップ オプションが選択されていれば、属性はデフォルト値で初期化されます。

例題

以下のコードは "Log" データクラスに新しいエンティティを作成し、"info" 属性に情報を記録します:

```
var $entity : cs.LogEntity
$entity:=ds.Log.new() // 参照を作成します
$entity.info:="New entry" // 情報を格納します
$entity.save() // エンティティを保存します
```

`.newSelection()`

▶ 覆歴

`.newSelection({ keepOrder : Integer }) : 4D.EntitySelection`

引数	タイプ		説明
keepOrder	整数	->	<code>dk keep ordered</code> : 順列ありのエンティティセレクションを作成します <code>dk non ordered</code> (あるいは省略時) : 順列なしのエンティティセレクションを作成します
戻り値	4D.EntitySelection	<-	データクラスの空の新規エンティティセレクション

説明

`.newSelection()` 関数は、データクラスに紐づいた 追加可能な、空の新規エンティティセレクションをメモリ内に作成します。

追加可能なエンティティセレクションについての詳細は [共有可能/追加可能なエンティティセレクション](#) を参照ください。

順列ありのエンティティセレクションを作成するには、`keepOrder` に `dk keep ordered` セレクターを渡します。この引数を省略した場合のデフォルト、あるいは `dk non ordered` セレクターを渡した場合には、関数は順列なしのエンティティセレクションを返します。順列なしのエンティティセレクションの方が速いですが、エンティティの位置に頼ることはできません。詳細については、[エンティティセレクションの順列あり/順列なし](#) を参照ください。

作成された時点では、エンティティセレクションにエンティティは含まれていません(`mySelection.length` は0を返します)。あとから `add()` 関数を呼び出すことで、エンティティセレクションを徐々にビルドしていくことができます。

例題

```
var $USelection; $OSelection : cs.EmployeeSelection  
$USelection:=ds.Employee.newSelection() // 順列なしの空のエンティティセレクションを作成します  
$OSelection:=ds.Employee.newSelection(dk keep ordered) // 順列ありの空のエンティティセレクションを作成します
```

.query()

▶履歴

.query(*queryString* : Text { ; ...*value* : any } { ; *querySettings* : Object }) : 4D.EntitySelection
.query(*formula* : Object { ; *querySettings* : Object }) : 4D.EntitySelection

引数	タイプ		説明
<i>queryString</i>	テキスト	->	検索条件 (文字列)
<i>formula</i>	オブジェクト	->	検索条件 (フォーミュラオブジェクト)
<i>value</i>	any	->	プレースホルダー用の値
<i>querySettings</i>	オブジェクト	->	クエリオプション: parameters, attributes, args, allowFormulas, context, queryPath, queryPlan
戻り値	4D.EntitySelection	<-	<i>queryString</i> または <i>formula</i> に渡した検索条件に合致するエンティティから構成された新しいエンティティセレクション

説明

.query() 関数は、データクラスの全エンティティから、*queryString* または *formula* と任意の *value* 引数で指定した検索条件に合致するエンティティを検索します。戻り値は、見つかったエンティティをすべて格納する EntitySelection 型の新しいオブジェクトです。この関数には、レイジーローディングが適用されます。

エンティティが見つからない場合、空のエンティティセレクションが返されます。

queryString 引数

queryString 引数には、以下のシンタックスを使用します:

```
attributePath|formula 比較演算子 値  
{論理演算子 attributePath|formula 比較演算子 値}  
{order by attributePath {desc | asc}}
```

詳細は以下の通りです:

- attributePath: クエリの実行対象となる属性パス。この引数は、単純な名前 ("country" など) のほか、あらゆる有効な属性パス ("country.name" など) の形をとることができます。属性パスが Collection 型である場合、すべてのオカレンスを管理するには[] 記法を使用してください (例: "children[].age" など)。

"!", "[]", や "=", ">", "#...", などの特殊文字はクエリ文字列の中で正しく評価されないため、これらが含まれた属性名を直接使用することはできません。このような属性をクエリするには、プレースホルダーの使用を検討します。これにより、属性パス内で使用できる文字の範囲が広がります (後述の プレースホルダーの使用 参照)。

- formula: テキストまたはオブジェクト形式で渡された有効なフォーミュラ。フォーミュラは処理されるエンティティごとに評価され、布尔値を返さなくてはなりません。処理中のエンティティはフォーミュラ内において This で参照されます。
 - テキスト: フォーミュラ文字列の前に eval() ステートメントが必要です。これにより、クエリが式を正しく解釈します。例: "eval(length(This.lastname) >=30)"
 - オブジェクト: フォーミュラオブジェクト は プレースホルダー (後述参照) を使って受け渡します。このフォーミュラは、Formula または Formula from string コマンドによって作成されたものでなくてはなりません。

- 4Dフォーミュラは、`&` および `|` 記号のみを論理演算子としてサポートすることに留意が必要です。
- フォーミュラ以外にも検索条件がある場合、クエリエンジンの最適化によってほかの検索条件（たとえばインデックス属性）の処理が優先される場合があり、その場合はエンティティのサブセットのみがフォーミュラの評価対象となります。

クエリに使用するフォーミュラは `$1` に引数を受け取ることができます。 詳細については後述の ****フォーミュラ引数**** を参照ください。

- フォーミュラが複雑な場合など、`queryString` パラメーターを使わずに、`formula` パラメーターにオブジェクトを直接渡すこともできます。後述の **フォーミュラ引数** を参照ください。
- セキュリティのため、`query()` 関数内のフォーミュラ使用を禁止することができます。`querySettings` パラメーターの説明を参照ください。

- 比較演算子: `attributePath` 引数と `value` 引数の比較に使用する記号 以下の記号がサポートされます:

比較	記号	説明
等しい	<code>=, ==</code>	一致するデータを取得します。ワイルドカード (@) をサポートし、文字の大小/アクセントの有無は区別しません。
	<code>====, IS</code>	一致するデータを取得します。ワイルドカード (@) は標準の文字として認識され、文字の大小/アクセント記号の有無は区別しません。
等しくない	<code>#, !=</code>	ワイルドカード (@) をサポートします
	<code>!=, IS NOT</code>	ワイルドカード (@) は標準の文字として認識されます
小さい	<code><</code>	
大きい	<code>></code>	
以下	<code><=</code>	
以上	<code>>=</code>	
含まれる	<code>IN</code>	コレクション、あるいは複数の値のうち、どれか一つの値と等しいデータを取得します。ワイルドカード (@) をサポートします。
宣言に Not 条件を適用	<code>NOT</code>	複数の演算子が含まれる宣言の前に NOT を使用する場合にはカッコをつける必要があります。
キーワードを含む	<code>%</code>	キーワードは、文字列あるいはピクチャー型の属性内で使用されるものが対象です。

- 値: コレクションの各要素、あるいはエンティティセレクションの各エンティティのプロパティのカレント値に対して比較する値。プレースホルダー（後述の **プレースホルダーの使用** 参照）か、あるいはデータ型プロパティと同じ型の式を使用することができます。

定数値を使用する場合、以下の原則に従う必要があります:

- テキスト テキスト型の定数値の場合は单一引用符つき、あるいはなしでも渡すことができます（後述の **引用符を使用する** 参照）。文字列中の文字列を検索する（"含まれる" クエリ）には、ワイルドカード記号 (@) を使用して検索文字列を指定します（例: "@Smith@")。また以下のキーワードはテキスト定数においては使用できません: `true`, `false`。
- ブール 型の定数値: `true` または `false` (文字の大小を区別します)
- 数値 型の定数値: 浮動小数点は `'.` (ピリオド) で区切られます。
- 日付 型の定数値: "YYYY-MM-DD" フォーマット。
- `null` 定数値: "null" キーワードを使用した場合、`null` と `undefined` プロパティの両方が検索されます。
- IN 記号を使用したクエリの場合、値 はコレクションか、`attributePath` の型に合致する、`[]` でくられたカンマ区切りの値である必要があります（文字列においては、" " の記号は \ でエスケープする必要があります）。
- 論理演算子: 複数の条件をクエリ内で結合させるのに使用します（任意）。以下の論理演算子のいずれか一つを使用できます（名前あるいは記号のどちらかを渡します）:

結合	記号
AND	&, &&, and
OR	, , or

- order by attributePath : クエリに "order by attributePath" ステートメントを追加することで、結果をソートすることができます。カンマで区切ることで、複数の order by ステートメントを使用することもできます (例: order by attributePath1 desc, attributePath2 asc)。デフォルトの並び順は昇順です。並び順を指定するには、降順の場合は 'desc'、昇順の場合は 'asc' を追加します。

このステートメントを使用した場合、順序ありエンティティセレクションが返されます (詳細については [エンティティセレクションの順列あり/順列なし](#) を参照ください)。

引用符を使用する

クエリ内で引用符を使用する場合、クエリ内においては単一引用符 ' ' を使用し、クエリ全体をくるには二重引用符 " " を使用します。クオートを混同するとエラーが返されます。たとえば:

```
"employee.name = 'smith' AND employee.firstname = 'john'"
```

単一引用符 (') は、クエリ文字列を分解してしまうため、検索値としてはサポートされていません。たとえば、"comp.name = 'John's pizza' " はエラーを生成します。単一引用符を含む値を検索するには、プレースホルダーを使用します (後述参照)。

カッコの使用

クエリ内でカッコを使用すると、計算に優先順位をつけることができます。たとえば、以下のようにクエリを整理することができます:

```
"(employee.age >= 30 OR employee.age <= 65) AND (employee.salary <= 10000 OR employee.status = 'Manager'
```

プレースホルダーの使用

4D では、*queryString* 引数内の *attributePath*、*formula* および 値 にプレースホルダーを使用することができます。プレースホルダーとは、クエリ文字列に挿入するパラメーターで、クエリ文字列が評価される時に他の値で置き換えられるものです。プレースホルダーの値はクエリ開始時に一度だけ評価されます。各要素に対して毎回評価されるわけではありません。

プレースホルダーには二つの種類があります。インデックスプレースホルダー および 命名プレースホルダー です:

	インデックスプレースホルダー	命名プレースホルダー
定義	<i>queryString</i> には、すべての種類の引数を混ぜて渡すことができます。 <i>queryString</i> 引数は、 <i>attributePath</i> と <i>formula</i> と 値 に以下のものを含めることができます:	:paramName (例: myparam など) という形でパラメーターが挿入され、その値は <i>querySettings</i> 引数の <i>attributes</i> または <i>parameters</i> オブジェクトで提供されます。
例題	\$r:=class.query(":1=:2";"city";"Chicago")	\$o.attributes:=New object("att";"city") \$o.parameters:=New object("name";"Chicago") \$r:=class.query(":att=:name";\$o)

queryString には、すべての種類の引数を混ぜて渡すことができます。*queryString* 引数は、*attributePath* と *formula* と 値 に以下のものを含めることができます:

- 定数値 (プレースホルダーを使用しない)
- インデックスプレースホルダーや命名プレースホルダー

以下の理由から、クエリでのプレースホルダーの使用が推奨されます:

- 悪意あるコードの挿入を防ぎます: ユーザーによって値が代入された変数をクエリ文字列として直接使用した場合、余計なクエリ引数を入力することでユーザーがクエリ条件を変更する可能性があります。たとえば、以下のようなクエリ文字列を考えます:

```
$vquery:="status = 'public' & name = "+myname // ユーザーが自分の名前を入力します  
$result:=$col.query($vquery)
```

非公開のデータがフィルタリングされているため、このクエリは一見安全なように見えます。しかしながら、もしユーザーが *myname* に *smith OR status='private'* のような入力をした場合、クエリ文字列は解釈時に変更され、非公開データも返してしまう可能性があります。

プレースホルダーを使用した場合、セキュリティ条件を上書きすることは不可能です：

```
$result:=$col.query("status='public' & name=:1";myname)
```

この場合、ユーザーが *myname* エリアに *smith OR status='private'* と入力した場合でも、それはクエリ文字列とはみなされず、値として渡されるだけです。*"smith OR status='private'"* という名前の人物を検索したところで、結果は失敗に終わるだけです。

2. フォーマットや文字の問題を心配する必要がありません。これは、*attributePath* や 値 がたとえば ".","[...などの英数字でない文字を格納している可能性がある場合にとくに有用です。
3. クエリに変数や式を使用することができます。例：

```
$result:=$col.query("address.city = :1 & name =:2;$city:$myVar+@")  
$result2:=$col.query("company.name = :1;"John's Pizzas")
```

null値の検索

null値を検索する場合、プレースホルダーシンタックスは使用できません。なぜならクエリエンジンは null を予期せぬ比較値としてみなすからです。たとえば、以下のクエリを実行した場合：

```
$vSingles:=ds.Person.query("spouse = :1;Null) // 機能しません
```

この場合 4D は null値を、引数の評価 (別のクエリから渡された属性など) に起因するエラーと解釈するため、期待した結果は得られません。このようないくつかの方法をおこなうには、直接的なシンタックスを使用する必要があります：

```
$vSingles:=ds.Person.query("spouse = null") // 正しいシンタックス
```

コレクション要素とクエリ条件のリンク

プロパティを複数持つオブジェクト要素からなりたっているコレクションが、さらに親オブジェクトの属性値である場合に、当該コレクションに特定要素が存在するかを条件に親オブジェクトを検出したいケースを考えます。AND 演算子で結合された複数のクエリ条件を使用して検索するだけでは、異なるコレクション要素がそれぞれ検索条件に合致するプロパティ値を持つ場合にも当該親オブジェクトが検出されてしまいます。これを避けるには、すべての条件に合致するコレクション要素のみが検出されるよう、クエリ条件をコレクション要素にリンクする必要があります。

たとえば、以下のような 2件のエンティティがあるとき：

```
エンティティ1:
ds.People.name: "martin"
ds.People.places:
{ "locations" : [ {
    "kind":"home",
    "city":"paris"
} ] }
```

```
エンティティ2:
ds.People.name: "smith"
ds.People.places:
{ "locations" : [ {
    "kind":"home",
    "city":"lyon"
} , {
    "kind":"office",
    "city":"paris"
} ] }
```

"locations" 属性に、"kind=home" かつ "city=paris" である要素を持つ人を探したいとします。以下のように書いた場合:

```
ds.People.query("places.locations[].kind= :1 and places.locations[].city= :2";"home";"paris")
```

... クエリは "martin" と "smith" の両方を返します。なぜなら "smith" も "kind=home" である "location" 要素を持っており、"city=paris" である "location" 要素をも持っているからです（ですがこれら 2つは異なる要素です）。

検索条件に合致する属性が同一のコレクション要素に含まれるエンティティのみを取得するには、クエリ条件をリンクします。クエリ条件をリンクするには:

- リンクする最初のクエリ条件にてバスの [] 内に文字を追加し、同様に他のクエリ条件でも同じ文字を追加します。例: locations[a].city and locations[a].kind。ローマ字であればどの文字でも使用可能です（文字の大小は区別されません）。
- 同じクエリ内に、異なるリンク条件を追加するには、別の文字を使用します。単一のクエリ内では、最大で 26 組のリンク条件を使用することができます。

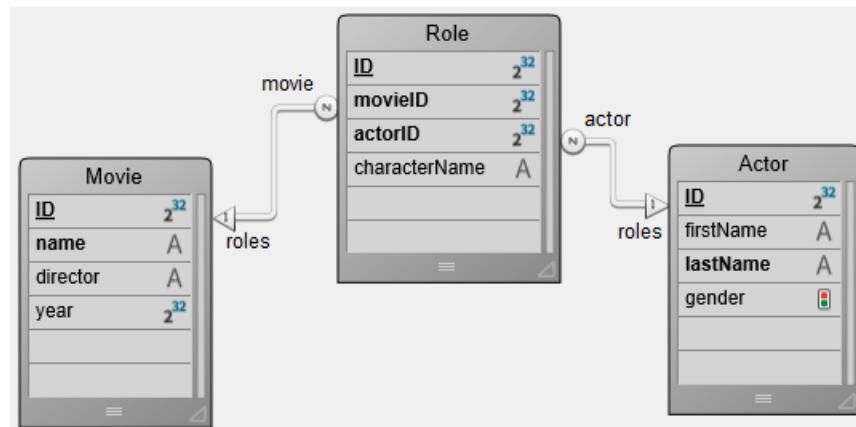
上記の 2 件のエンティティにおいて、以下のように書いた場合:

```
ds.People.query("places.locations[a].kind= :1 and places.locations[a].city= :2";"home";"paris")
```

... クエリは、"kind=home" かつ "city=paris" である "locations" 要素を持つエンティティ "martin" のみを返します。"home" と "paris" が同じコレクション要素内にない "smith" は、クエリの結果に含まれません。

N対Nリレーションのクエリ

ORDA は、N対Nリレーションにおけるクエリを容易にするための特別な構文を提供します。このような場合には、AND 演算子を使って同じ属性の異なる値を検索する必要があるかもしれません。たとえば、以下のようなストラクチャーの場合を考えます:



俳優 A と俳優 B の両方 が出演している映画をすべて検索したいとします。AND 演算子を使った単純なクエリを書いても、うまくいきません:

```
// 無効なコード  
$es:=ds.Movie.query("roles.actor.lastName = :1 AND roles.actor.lastName = :2","Hanks","Ryan")  
// $es is empty
```

基本的に、この問題はクエリの内部ロジックに関連しています。値が "A" と "B" の両方である属性を検索することはできません。

比較演算子: *attributePath* 引数と *value* 引数の比較に使用する記号 以下の記号がサポートされます:

```
"relationAttribute.attribute = :1 AND relationAttribute{x}.attribute = :2 [AND relationAttribute{y}.attr
```

{x} は、リレーション属性のために新たな参照を作成するよう、ORDA に指示します。すると、ORDA は必要なビットマップ操作を内部で実行します。x は 0 以外 の任意の数であることに注意してください ({1}, {2}, {1540}...)。ORDA は、各クラスインデックス用の一意な参照をクエリ内においてのみ必要とします。

この例では、次のようになります:

```
// 有効なコード  
$es:=ds.Movie.query("roles.actor.lastName = :1 AND roles.actor{2}.lastName = :2","Hanks","Ryan")  
// $es には映画が格納されます (You've Got Mail, Sleepless in Seattle, Joe Versus the Volcano)
```

formula 引数

queryString 引数にフォーミュラを挿入 (上記参照) する代わりに、formula オブジェクトをブール検索条件として直接渡すことができます。トークナイズの利点を生かせる、コードが検索しやすく読みやすい、などといった面から、クエリにおけるフォーミュラオブジェクトの使用は 推奨されています。

このフォーミュラは、[Formula](#) または [Formula from string](#) コマンドによって作成されたものでなくてはなりません。この場合において:

- フォーミュラは処理されるエンティティごとに評価され、true または false を返さなければなりません。クエリの実行中、フォーミュラの結果がブール値でなかった場合、それは false であるとみなされます。
- 処理中のエンティティはフォーミュラ内において `This` で参照されます。
- `Formula` オブジェクトが null の場合、エラー1626 ("テキストまたはフォーミュラが必要です") が生成されます。

セキュリティのため、`query()` 関数内のフォーミュラ使用を禁止することができます。`querySettings` パラメーターの説明を参照ください。

フォーミュラに引数を渡す

`query()` クラス関数によって呼び出されるフォーミュラは、引数を受け取ることができます:

- 引数は、`querySettings` 引数の `args` プロパティ (オブジェクト) を通して渡さなければなりません。
- フォーミュラは `args` オブジェクトを `$1` に受け取ります。

以下の短いコードは、引数をフォーミュラに渡す仕組みを示しています:

```
$settings:=New object("args";New object("exclude";"-")) // 引数を渡すための args オブジェクト  
$es:=ds.Students.query("eval(checkName($1.exclude))";$settings) // $1 が args を受け取ります
```

さらなる使用例は、例題3にて紹介されています。

4D Server: クライアント/サーバーにおいては、フォーミュラはサーバー上で実行されます。このコンテキストにおいては、`querySettings.args` オブジェクトのみがフォーミュラに送信されます。

querySettings 引数

`querySettings` 引数は、追加のオプションを格納したオブジェクトです。以下のオブジェクトプロパティがサポートされています:

プロパティ	タイプ	説明						
parameters	オブジェクト	<i>queryString</i> または <i>formula</i> に 値の命名プレースホルダー を使用した場合に渡すオブジェクト。値は、プロパティ/値のペアで表現されます。プロパティは、 <i>queryString</i> または <i>formula</i> に値の代わりに挿入されたプレースホルダー名 ("":placeholder"など) で、値は、実際に比較される値です。インデックスプレースホルダー (value引数として値を直接渡す方法) と命名プレースホルダーは、同じクエリ内で同時に使用することができます。						
attributes	オブジェクト	<i>queryString</i> または <i>formula</i> に 属性パスの命名プレースホルダー を使用した場合に渡すオブジェクト。属性パスは、プロパティ/値のペアで表現されます。プロパティは、 <i>queryString</i> または <i>formula</i> に属性パスの代わりに挿入されたプレースホルダー名 ("":placeholder"など) で、値は、属性パスを表す文字列または文字列のコレクションです。値には、データクラスのスカラー属性・リレート属性・オブジェクトフィールド内のプロパティへの属性パスを指定することができます。 <table border="1" style="margin-left: 20px;"> <thead> <tr> <th>タイプ</th><th>説明</th></tr> </thead> <tbody> <tr> <td>String</td><td>ドット記法を使用して表現された attributePath (例: "name" または "user.address.zipCode")</td></tr> <tr> <td>文字列のコレクション</td><td>コレクションの各要素が attributePath の階層を表します (例: ["name"] または ["user","address","zipCode"])。コレクションを使用することで、ドット記法に準じていない名前の属性に対してもクエリすることができます (例: ["4Dv17.1","en/fr"])。</td></tr> </tbody> </table>	タイプ	説明	String	ドット記法を使用して表現された attributePath (例: "name" または "user.address.zipCode")	文字列のコレクション	コレクションの各要素が attributePath の階層を表します (例: ["name"] または ["user","address","zipCode"])。コレクションを使用することで、ドット記法に準じていない名前の属性に対してもクエリすることができます (例: ["4Dv17.1","en/fr"])。
タイプ	説明							
String	ドット記法を使用して表現された attributePath (例: "name" または "user.address.zipCode")							
文字列のコレクション	コレクションの各要素が attributePath の階層を表します (例: ["name"] または ["user","address","zipCode"])。コレクションを使用することで、ドット記法に準じていない名前の属性に対してもクエリすることができます (例: ["4Dv17.1","en/fr"])。							
		インデックスプレースホルダー (value 引数として値を直接渡す方法) と命名プレースホルダーは、同じクエリ内で同時に使用することができます。						
args	オブジェクト	フォーミュラに渡す引数。args オブジェクトは、フォーミュラ内の \$1 が受け取るので、その値は \$1.property という形で利用可能です (例題3 参照)。						
allowFormulas	布尔	クエリ内でフォーミュラの呼び出しを許可するには true (デフォルト)。フォーミュラ実行を禁止するには false を渡します。false に設定されているときに、フォーミュラが query() に渡された場合、エラーが発生します (1278 - フォーミュラはこのメンバーメソッドでは許可されません)。						
context	テキスト	エンティティセレクションに適用されている自動の最適化コンテキストのラベル。エンティティセレクションを扱うコードはこのコンテキストを使うことで最適化の恩恵を受けます。この機能はクライアント/サーバー処理を想定して設計されています。詳細な情報については、 クライアント/サーバーの最適化 の章を参照ください。						
queryPlan	布尔	戻り値のエンティティコレクションに、実行する直前のクエリの詳細 (クエリプラン) を含めるかどうかを指定します。返されるプロパティは、クエリプラン あるいは サブクエリ (複合クエリの場合) を格納したオブジェクトです。このオプションはアプリケーションの開発フェーズにおいて有用です。このオプションは通常 queryPath と組み合わせて使用されます。省略時のデフォルト: false。注: このプロパティは entitySelection.query() および dataClass.query() 関数においてのみサポートされます。						
queryPath	布尔	戻り値のエンティティコレクションに、実際に実行されたクエリの詳細を含めるかどうかを指定します。返されたプロパティは、クエリで実際に使用されたパス (通常は queryPlan と同一ですが、エンジンがクエリを最適化した場合には異なる場合があります)、処理時間と検出レコード数を格納したオブジェクトです。このオプションはアプリケーションの開発フェーズにおいて有用です。省略時のデフォルト: false。注: このプロパティは entitySelection.query() および dataClass.query() 関数においてのみサポートされます。						

queryPlan と queryPath について

queryPlan / queryPath に格納される情報には、クエリの種類 (インデックスあるいはシーケンシャル)、必要なサブクエリおよびその連結演算子が含まれます。クエリパスには、見つかったエンティティの数と各検索条件を実行するにかかる時間も含まれます。この情報は、アプリケーションの開発中に解析することで有効に活用できます。一般的には、クエリプランとクエリパスの詳細は同一になるはずですが、4D はパフォーマンスの向上のために、動的な最適化をクエリ実行時に実装することがあるからです。たとえば、その方が早いと判断した場合には、4Dエンジンはインデックス付きクエリをシーケンシャルなものへと動的に変換することができます。これは検索されているエンティティの数が少ないときに起こります。

たとえば、以下のクエリを実行した場合:

```
$sel:=ds.Employee.query("salary < :1 and employer.name = :2 or employer.revenues > :3";\n50000;"Lima West Kilo";10000000;New object("queryPath";True;"queryPlan";True))
```

queryPlan:

```
{Or:[{And:[{item:[index : Employee.salary ] < 50000},  
    {item:Join on Table : Company : Employee.employerID = Company.ID,  
    subquery:[{item:[index : Company.name ] = Lima West Kilo}]}]},  
    {item:Join on Table : Company : Employee.employerID = Company.ID,  
    subquery:[{item:[index : Company.revenues ] > 10000000}]}]}
```

queryPath:

```
{steps:[{description:OR,time:63,recordsfounds:1388132,  
    steps:[{description:AND,time:32,recordsfounds:131,  
        steps:[{description:[index : Employee.salary ] < 50000,time:16,recordsfounds:728260},{description:Jo  
            steps:[{steps:[{description:[index : Company.name ] = Lima West Kilo,time:0,recordsfounds:1}]}]}]},  
            steps:[{steps:[{description:[index : Company.revenues ] > 10000000,time:0,recordsfounds:933}]}]}]}
```

例題 1

この例題では、様々なクエリの例を紹介します。

文字列のクエリ:

```
$entitySelection:=ds.Customer.query("firstName = 'S@'")
```

NOT節を用いたクエリ:

```
$entitySelection:=ds.Employee.query("not(firstName=Kim)")
```

日付のクエリ:

```
$entitySelection:=ds.Employee.query("birthDate > :1";"1970-01-01")  
$entitySelection:=ds.Employee.query("birthDate <= :1";Current date-10950)
```

値のインデックスプレースホルダーを使用したクエリ:

```
$entitySelection:=ds.Customer.query("(firstName = :1 or firstName = :2) and (lastName = :3 or lastName = :4);(:1,:2,:3,:4)" ; ("John","Mike","Doe","Doe"))
```

リレートデータクラスに対して値のインデックスプレースホルダーを使用したクエリ:

```
$entitySelection:=ds.Employee.query("lastName = :1 and manager.lastName = :2";"M@";"S@")
```

降順の order by ステートメントを含んだ、インデックスプレースホルダーを使用したクエリ:

```
$entitySelection:=ds.Student.query("nationality = :1 order by campus.name desc, lastname";"French")
```

値の命名プレースホルダーを使用したクエリ:

```
var $querySettings : Object
var $managedCustomers : cs.CustomerSelection
$querySettings:=New object
$querySettings.parameters:=New object("userId";1234;"extraInfo";New object("name";"Smith"))
$managedCustomers:=ds.Customer.query("salesperson.userId = :userId and name = :extraInfo.name";$querySet
```

値の命名プレースホルダーと、値のインデックスプレースホルダーの両方を使用したクエリ:

```
var $querySettings : Object
var $managedCustomers : cs.CustomerSelection
$querySettings.parameters:=New object("userId";1234)
$managedCustomers:=ds.Customer.query("salesperson.userId = :userId and name=:1";"Smith";$querySettings)
```

queryPlan および queryPath オブジェクトを返すクエリ:

```
$entitySelection:=ds.Employee.query("(firstName = :1 or firstName = :2) and (lastName = :3 or lastName = :4);"
// 戻り値のエンティティセレクションから以下のプロパティを取得できます
var $queryPlan; $queryPath : Object
$queryPlan:=$entitySelection.queryPlan
$queryPath:=$entitySelection.queryPath
```

コレクション型の属性パスを用いたクエリ:

```
$entitySelection:=ds.Employee.query("extraInfo.hobbies[].name = :1";"horsebackriding")
```

コレクション型の属性パスとクエリ条件をリンクしたクエリ:

```
$entitySelection:=ds.Employee.query("extraInfo.hobbies[a].name = :1 and extraInfo.hobbies[a].level=:2";"
```

コレクション型の属性パスとクエリ条件を、複数リンクしたクエリ:

```
$entitySelection:=ds.Employee.query("extraInfo.hobbies[a].name = :1 and
extraInfo.hobbies[a].level = :2 and extraInfo.hobbies[b].name = :3 and
extraInfo.hobbies[b].level = :4";"horsebackriding";2;"Tennis";5)
```

オブジェクト型の属性パスを用いたクエリ:

```
$entitySelection:=ds.Employee.query("extra.eyeColor = :1";"blue")
```

IN節を用いたクエリ:

```
$entitySelection:=ds.Employee.query("firstName in :1";New collection("Kim";"Dixie"))
```

NOT (IN) 節を用いたクエリ:

```
$entitySelection:=ds.Employee.query("not (firstName in :1)";New collection("John";"Jane"))
```

属性パスのインデックスプレースホルダーを使用したクエリ:

```
var $es : cs.EmployeeSelection
$es:=ds.Employee.query(":1 = 1234 and :2 = 'Smith'";"salesperson.userId";"name")
// salesperson はリレートエンティティです
```

属性パスのインデックスプレースホルダーを、値の命名プレースホルダーを使用したクエリ:

```
var $es : cs.EmployeeSelection
var $querySettings : Object
$querySettings:=New object
$querySettings.parameters:=New object("customerName";"Smith")
$es:=ds.Customer.query(":1 = 1234 and :2 = :customerName";"salesperson.userId";"name";$querySettings)
// salesperson はリレートエンティティです
```

属性パスと値のインデックスプレースホルダーを使用したクエリ:

```
var $es : cs.EmployeeSelection
$es:=ds.Clients.query(":1 = 1234 and :2 = :3";"salesperson.userId";"name";"Smith")
// salesperson はリレートエンティティです
```

例題 2

この例題では、属性パスの命名プレースホルダーを使用するクエリを紹介します。

2件のエンティティをもつ Employee データクラスを前提に考えます:

エンティティ1:

```
name: "Marie"
number: 46
softwares:[
  "Word 10.2": "Installed",
  "Excel 11.3": "To be upgraded",
  "Powerpoint 12.4": "Not installed"
}
```

エンティティ2:

```
name: "Sophie"
number: 47
softwares:[
  "Word 10.2": "Not installed",
  "Excel 11.3": "To be upgraded",
  "Powerpoint 12.4": "Not installed"
}
```

属性パスの命名プレースホルダーを使用したクエリ:

```
var $querySettings : Object
var $es : cs.EmployeeSelection
$querySettings:=New object
$querySettings.attributes:=New object("attName";"name";"attWord";New collection("softwares";"Word 10.2")
$es:=ds.Employee.query(":attName = 'Marie' and :attWord = 'Installed'";$querySettings)
// $es.length=1 (Employee Marie)
```

属性パスと値の命名プレースホルダーを使用したクエリ:

```
var $querySettings : Object
var $es : cs.EmployeeSelection
var $name : Text
$querySettings:=New object
// 値の命名プレースホルダー
// ユーザーに検索する名前を入力してもらいます
$name:=Request("検索する名前を入力してください:")
If(OK=1)
    $querySettings.parameters:=New object("givenName";$name)
// 属性パスの命名プレースホルダー
$querySettings.attributes:=New object("attName";"name")
$es:=ds.Employee.query(":attName= :givenName";$querySettings)
End if
```

例題 3

この例題では、クエリにおいて、引数あり・引数なしでフォーミュラを使用する様々な方法を紹介します。

`eval()` を使い、テキストとしてフォーミュラを *queryString* パラメーターに渡すクエリ:

```
var $es : cs.StudentsSelection
$es:=ds.Students.query("eval(length(This.lastname) >=30) and nationality='French'")
```

プレースホルダーを使い、`Formula` オブジェクトとしてフォーミュラを渡すクエリ:

```
var $es : cs.StudentsSelection
var $formula : Object
$formula:=Formula(Length(This.lastname)>=30)
$es:=ds.Students.query(":1 and nationality='French'";$formula)
```

`Formula` オブジェクトのみを検索条件として渡したクエリ:

```
var $es : cs.StudentsSelection
var $formula : Object
$formula:=Formula(Length(This.lastname)>=30)
$es:=ds.Students.query($formula)
```

フォーミュラを複数適用したクエリ:

```
var $formula1; $1; $formula2 ;$0 : Object
$formula1:=$1
$formula2:=Formula(Length(This.firstname)>=30)
$0:=ds.Students.query(":1 and :2 and nationality='French'";$formula1;$formula2)
```

queryString のテキストフォーミュラが引数を受け取るクエリ:

```
var $es : cs.StudentsSelection
var $settings : Object
$settings:=New object()
$settings.args:=New object("filter";"-")
$es:=ds.Students.query("eval(checkName($1.filter)) and nationality=:1";"French";$settings)
```

```
// checkName メソッド
#DECLARE($exclude : Text) -> $result : Boolean
$result:=(Position($exclude;This.lastname)=0)
```

同じ `checkName` メソッドを `Formula` オブジェクトに格納してプレースホルダーで渡し、引数を受け取るクエリ:

```
var $es : cs.StudentsSelection
var $settings; $formula : Object
$formula:=Formula(checkName($1.filter))
$settings:=New object()
$settings.args:=New object("filter";"-")
$es:=ds.Students.query(":1 and nationality=:2;$formula;"French";$settings)
$settings.args.filter:="*" // $formula オブジェクトは更新せずに引数を変更します
$es:=ds.Students.query(":1 and nationality=:2;$formula;"French";$settings)
```

ユーザーがクエリを入力する場合などに、フォーミュラを禁止する場合:

```
var $es : cs.StudentsSelection
var $settings : Object
var $queryString : Text
$queryString:=Request("Enter your query:")
if(OK=1)
    $settings:=New object("allowFormulas";False)
    $es:=ds.Students.query($queryString;$settings) // $queryString にフォーミュラが格納されていた場合にはエラーが発生する
End if
```

参照

エンティティセレクションの `.query()`

.setRemoteCacheSettings()

▶ 履歴

`.setRemoteCacheSettings(settings : Object)`

引数	タイプ		説明
settings	オブジェクト	->	データクラスの ORDAキャッシュについて、タイムアウトと最大サイズを指定するオブジェクト

上級者向け: この機能は、特定の構成のため、ORDAのデフォルト機能をカスタマイズする必要がある開発者向けです。ほとんどの場合、使用する必要はないでしょう。

説明

`.setRemoteCacheSettings()` 関数は、データクラスの ORDAキャッシュについて、タイムアウトと最大サイズを指定します。

`settings` には、以下のプロパティを持つオブジェクトを渡します:

プロパティ	タイプ	説明
timeout	整数	タイムアウト (秒単位)
maxEntries	整数	エンティティの最大数

`timeout` は、データクラスの ORDAキャッシュのタイムアウトを設定します (デフォルトは 30秒)。タイムアウトを過ぎると、キャッシング内のデータクラスのエンティティは期限切れとみなされます。これは、次のことを意味します:

- データはまだ存在します
- 次にそのデータが必要になったときには、サーバーに要求します
- エンティティの最大数に達すると、4D は期限切れのデータを自動的に削除します

`timeout` プロパティを設定すると、すでにキャッシュに存在するエンティティに新しいタイムアウトが設定されます。これは頻繁に変更されないデータを扱う場合、つまり、サーバーへの新たな要求が必要ない場合に便利です。

`maxEntries` は、ORDAキャッシュ内のエンティティの最大数を設定します。デフォルトは 30,000 です。

最小エントリー数は 300 のため、`maxEntries` の値は 300以上でなくてはなりません。それ以外の場合は無視され、最大エントリー数は 300 に設定されます。

`timeout` および `maxEntries` として有効なプロパティが渡されない場合、キャッシュはデフォルト値または以前に設定された値のまま変更されません。

エンティティが保存されると、キャッシュ内で更新され、タイムアウトに達すると期限切れとなります。

例題

```
var $ds : 4D.DataStoreImplementation  
  
$ds:=Open datastore(New object("hostname"; "www.myserver.com"); "myDS")  
  
$ds.Buildings.setRemoteCacheSettings(New object("timeout"; 60; "maxEntries"; 350))
```

参照

[.clearRemoteCache\(\)](#)
[.getRemoteCache\(\)](#)

DataClassAttribute

データクラス属性は、それぞれのクラスのプロパティとして利用可能です。たとえば:

```
nameAttribute:=ds.Company.name // クラス属性への参照  
revenuesAttribute:=ds.Company["revenues"] // 別の書き方
```

このコードは、`nameAttribute` および `revenuesAttribute` 変数に、`Company` クラスの `name` および `revenues` 属性の参照をそれぞれ代入します。このシンタックスは属性内に保管されている値を返すではありません。その代わりに、属性自身への参照を返します。値を管理するには、[エンティティ](#) を使用する必要があります。

`DataClassAttribute` オブジェクトが持つプロパティを読み取ることによって、データクラス属性に関する情報が取得できます。

データクラス属性オブジェクトを編集することは可能ですが、元となるデータベースストラクチャーは変更されません。

概要

<code>.autoFilled : Boolean</code>	属性値が 4Dによって自動生成される場合に true
<code>.exposed : Boolean</code>	属性が REST で公開されている場合に true
<code>.fieldNumber : Integer</code>	属性の内部的な 4Dフィールド番号
<code>.fieldType : Integer</code>	属性の 4Dデータベースタイプ
<code>.indexed : Boolean</code>	属性に対して B-tree もしくは クラスターB-Tree インデックスが設定されている場合に true
<code>.inverseName : Text</code>	リレーション先の属性名
<code>.keywordIndexed : Boolean</code>	属性にキーワードインデックスが存在すれば true
<code>.kind : Text</code>	属性の種類
<code>.mandatory : Boolean</code>	属性において Null値の入力が拒否されている場合に true
<code>.name : Text</code>	<code>dataClassAttribute</code> オブジェクトの名称
<code>.path : Text</code>	リレーション属性を指し示すエイリアス属性のパス
<code>.readOnly : Boolean</code>	読み取り専用属性の場合に true
<code>.relatedDataClass : Text</code>	属性にリートされているデータクラスの名称
<code>.type : Text</code>	属性の概念的な値タイプ
<code>.unique : Boolean</code>	属性値が重複不可の場合に true

.autoFilled

▶ 履歴

`.autoFilled : Boolean`

説明

`.autoFilled` プロパティは、属性値が 4Dによって自動生成される場合に trueです。このプロパティは以下の 4Dフィールドプロパティに対応しています:

- 数値型フィールドの "自動インクリメント"

- UUID (文字型)フィールドの "自動UUID"

.kind が "relatedEntity" または "relatedEntities" の場合には、このプロパティは返されません。

汎用的なプログラミングのために、.autoFilled が返されない場合でも Bool (dataClassAttribute.autoFilled) と書くことで、有効な値 (false) を受け取ることができます。

.exposed

▶履歴

.exposed : Boolean

説明

.exposed プロパティは、属性が REST で公開されている場合に trueです。

参照

[DataClass.getInfo\(\)](#)

.fieldNumber

▶履歴

.fieldNumber : Integer

説明

.fieldNumber プロパティは、属性の内部的な 4Dフィールド番号を格納します。

.kind が "relatedEntity" または "relatedEntities" の場合には、このプロパティは返されません。

汎用的なプログラミングのために、.fieldNumber が返されない場合でも Num (dataClassAttribute.fieldNumber) と書くことで、有効な値 (0) を受け取ることができます。

.fieldType

▶履歴

.fieldType : Integer

説明

.fieldType プロパティは、属性の 4Dデータベースタイプを格納します。これは属性の種類 (kind) によります (.kind 参照)。

とりうる値:

dataClassAttribute.kind	fieldType
storage	4Dフィールドタイプに対応、 Value type コマンド参照
relatedEntity	38 (Is object)
relatedEntities	42 (Is collection)
calculated	<ul style="list-style-type: none"> スカラー: 4Dフィールドタイプに対応、Value type コマンド参照 エンティティ: 38 (Is object) エンティティセレクション: 42 (Is collection)
alias	<ul style="list-style-type: none"> スカラー: 4Dフィールドタイプに対応、Value type コマンド参照 エンティティ: 38 (Is object) エンティティセレクション: 42 (Is collection)

参照

[.type](#)

.indexed

▶履歴

.indexed : Boolean

説明

`.indexed` プロパティは、属性に対して B-tree もしくは クラスターB-Tree インデックスが設定されている場合に trueです。

`.kind` が "relatedEntity" または "relatedEntities" の場合には、このプロパティは返されません。

汎用的なプログラミングのために、`.indexed` が返されない場合でも Bool (dataClassAttribute.indexed) と書くことで、有効な値 (false) を受け取ることができます。

.inverseName

▶履歴

.inverseName : Text

説明

`.inverseName` プロパティは、リレーション先の属性名を格納します。

`.kind` が "storage" の場合には、このプロパティは返されません。.kind は "relatedEntity" または "relatedEntities" でなくてはなりません。

汎用的なプログラミングのために、`.inverseName` が返されない場合でも String (dataClassAttribute.inverseName) と書くことで、有効な値 ("") を受け取ることができます。

.keywordIndexed

▶履歴

.keywordIndexed : Boolean

説明

`.keywordIndexed` プロパティは、属性にキーワードインデックスが存在すれば trueです。

.kind が "relatedEntity" または "relatedEntities" の場合には、このプロパティは返されません。

汎用的なプログラミングのために、.keywordIndexed が返されない場合でも Bool (dataClassAttribute.keywordIndexed) と書くことで、有効な値 (false) を受け取ることができます。

.kind

▶ 補足

.kind : Text

説明

.kind プロパティは、属性の種類を格納します。以下のいずれかの値が返されます:

- "storage": ストレージ (あるいはスカラー) 属性。つまり、属性は値を保存しており、他の属性への参照ではありません。
- "calculated": 計算属性。get 関数 によって定義されます。
- "alias": 他の属性 を指示する属性。
- "relatedEntity": N対1 リレーション属性 (エンティティへの参照)
- "relatedEntities": 1対N リレーション属性 (エンティティセレクションへの参照)

例題

以下のテーブルとリレーションを前提とします:

```
var $attKind : Text
$attKind:=ds.Employee.lastname.kind // $attKind="storage"
$attKind:=ds.Employee.manager.kind // $attKind="relatedEntity"
$attKind:=ds.Employee.directReports.kind // $attKind="relatedEntities"
```

.mandatory

▶ 補足

.mandatory : Boolean

説明

.mandatory プロパティは、属性において Null 値の入力が拒否されている場合に true です。

.kind が "relatedEntity" または "relatedEntities" の場合には、このプロパティは返されません。

汎用的なプログラミングのために、.mandatory が返されない場合でも Bool (dataClassAttribute.mandatory) と書くことで、有効な値 (false) を受け取ることができます。警告: このプロパティは、4Dデータベースレベルの "Null 値の入力を拒否" フィールドプロパティと対応しています。フィールドのデータ入力制御オプションである既存の "必須入力" プロパティとは無関係です。

.name

▶ 補足

.name : Text

説明

.name プロパティは、dataClassAttribute オブジェクトの名称を文字列として返します。

例題

```
var $attName : Text  
$attName:=ds.Employee.lastname.name // $attName="lastname"
```

.path

▶ 履歴

.path : Text

説明

.path プロパティは、リレーション属性を指し示すエイリアス属性のパスを返します。

例題

```
var $path : Text  
$path:=ds.Teacher.students.path // $path="courses.student"
```

.readOnly

▶ 履歴

.readOnly : Boolean

説明

.readOnly プロパティは、読み取り専用属性の場合に trueです。

たとえば、`set 関数`を持たない計算属性は読み取り専用です。

.relatedDataClass

▶ 履歴

.relatedDataClass : Text

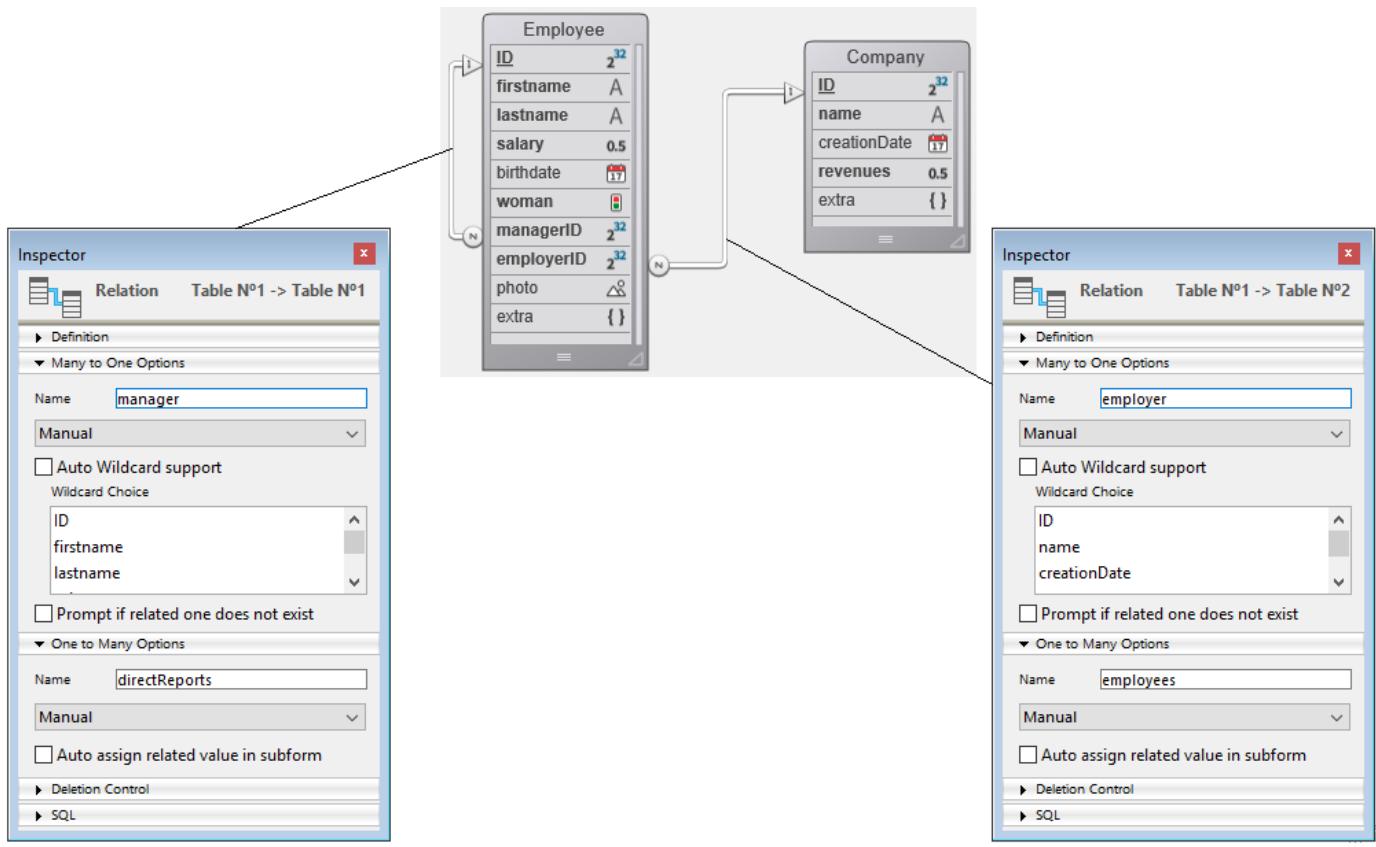
説明

このプロパティは、.kind プロパティ値が "relatedEntity" または "relatedEntities" である属性においてのみ利用可能です。

.relatedDataClass プロパティ、属性にリレートされているデータクラスの名称を返します。

例題

以下のテーブルとリレーションを前提とします:



```
var $relClass1; $relClassN : Text
$relClass1:=ds.Employee.employer.relatedDataClass // $relClass1="Company"
$relClassN:=ds.Employee.directReports.relatedDataClass // $relClassN="Employee"
```

.type

▶ 履歴

.type : Text

説明

.type プロパティは、属性の概念的な値タイプが格納されており、汎用的なプログラミングに有用です。

この概念的な値タイプは属性の種類 (.kind) によります。

とりうる値:

dataClassAttribute.kind	type	説明
storage	"blob", "bool", "date", "image", "number", "object", または "string"	数値型の場合 "number" が返されます (期間を含む)。UUID、文字およびテキスト型フィールドの場合 "string" が返されます。"blob" 属性は BLOB オブジェクト で、 Blob クラス によって扱われます。
relatedEntity	リレートされたデータクラス名	例: "Companies"
relatedEntities	リレートされたデータクラス名 + "Selection"	例: "EmployeeSelection"
calculated	<ul style="list-style-type: none"> ● ストレージ: データ型 ("blob", "number", など) ● エンティティ: データクラス名 ● エンティティセレクション: データクラス名 + "Selection" 	

参照

[.fieldType](#)

.unique

▶ 覆歴

.unique : Boolean

説明

.unique プロパティは、属性値が重複不可の場合に trueです。このプロパティは、4Dフィールドプロパティの "重複不可" に対応しています。

[.kind](#) が "relatedEntity" または "relatedEntities" の場合には、このプロパティは返されません。

汎用的なプログラミングのために、.unique が返されない場合でも Bool (dataClassAttribute.unique) と書くことで、有効な値 (false) を受け取ることができます。

DataStore

データストアとは、ORDAによって提供されるインターフェースオブジェクトです。データストアはデータベースへの参照とアクセスを提供します。

Datastore オブジェクトは以下のコマンドによって返されます:

- [ds](#): メインデータストアへのショートカット
- [Open datastore](#): リモートデータストアを開きます

概要

.cancelTransaction() トランザクションをキャンセルします
.dataclassName : 4D.DataClass データクラスの詳細が格納されています
.encryptionStatus(): Object カレントデータファイルの暗号化状態を示すオブジェクトを返します
.getInfo(): Object データストアの情報を提供するオブジェクトを返します
.getRequestLog() : Collection クライアント側のメモリに記録されている ORDAリクエストを返します
.makeSelectionsAlterable() カレントアプリケーションのデータストアにおいて、すべての新規エンティティセレクションをデフォルトで追加可能に設定します
.provideDataKey(curPassPhrase : Text) : Object .provideDataKey(curDataKey : Object) : Object データストアのカレントデータファイルのデータ暗号化キーを受け取り、暗号化されたデータと合致するかどうかチェックします
.setAdminProtection(status : Boolean) WebAdmin セッションにおける データエクスプローラー 含め、Web管理ポート 上でのデータアクセスを無効に設定することができます
.startRequestLog() .startRequestLog(file : 4D.File) .startRequestLog(reqNum : Integer) クライアント側で ORDAリクエストのログを開始します
.startTransaction() 対象データストアに対応するデータベース上で、カレントプロセス内のトランザクションを開始します
.stopRequestLog() クライアント側の ORDAリクエストのログをすべて停止します
.validateTransaction() トランザクションを受け入れます

ds

▶履歴

ds { (localID : Text) } : cs.DataStore

引数	タイプ		説明
localID	Text	->	参照を取得したいリモートデータストアのローカルID
戻り値	cs.DataStore	<-	データストア参照

説明

`ds` コマンドは、カレントの 4Dデータベース、または `localID` で指定したデータベースに合致するデータストアの参照を返します。

`localID` を省略した（または空の文字列 "" を渡した）場合には、ローカル4Dデータベース（4D Server でリモートデータベースを開いている場合にはそのデータベース）に合致するデータストアの参照を返します。データストアは自動的に開かれ、`ds` を介して直接利用することができます。

開かれているリモートデータストアのローカルIDを `localID` パラメーターに渡すと、その参照を取得できます。このデータストアは、あらかじめカレントデータベース（ホストまたはコンポーネント）によって `Open datastore` コマンドで開かれている必要があります。このコマンドを使用したときにローカルIDが定義されます。

ローカルIDのスコープは、当該データストアを開いたデータベースです。

`localID` に合致するデータストアが見つからない場合、コマンドは Null を返します。

`cs.Datastore` が提供するオブジェクトは、[ORDAマッピングルール](#) に基づいて、ターゲットデータベースからマッピングされます。

例題 1

4Dデータベースのメインデータストアを使用します：

```
$result:=ds.Employee.query("firstName = :1";"S@")
```

例題 2

```
var $connectTo; $firstFrench; $firstForeign : Object
var $frenchStudents; $foreignStudents : cs.DataStore
$connectTo:=New object("type";"4D Server";"hostname";"192.168.18.11:8044")
$frenchStudents:=Open datastore($connectTo;"french")

$connectTo.hostname:="192.168.18.11:8050"
$foreignStudents:=Open datastore($connectTo;"foreign")
//...
//...
$firstFrench:=getFirst("french";"Students")
$firstForeign:=getFirst("foreign";"Students")
```

```
// getFirst メソッド
// getFirst(localID;dataclass) -> entity
#DECLARE( $localId : Text; $dataClassName : Text ) -> $entity : 4D.Entity
$0:=ds($localId)[$dataClassName].all().first()
```

Open datastore

▶ 覆歴

`Open datastore(connectionInfo : Object ; localID : Text) : cs.DataStore`

引数	タイプ		説明
connectionInfo	Object	->	リモートデータストアへの接続に使用する接続プロパティ
localID	Text	->	ローカルアプリケーション内で、開かれたデータストアに対して割り当てる ID (必須)
戻り値	cs.DataStore	<-	データストアオブジェクト

説明

`Open datastore` コマンドは、`connectionInfo` 引数が指定する 4Dデータベースにアプリケーションを接続します。戻り値は、`localID` ローカルエイリアスに紐づけられた `cs.DataStore` オブジェクトです。

`connectionInfo` で指定する 4Dデータベースはリモートデータストアとして利用可能でなければなりません。つまり、以下の条件を満たしている必要があります:

- データベースの Webサーバーは、http または https が有効化された状態で開始されていなければなりません。
- データベースの
- データベースにおいて、少なくとも 1つのクライアントライセンスが利用可能でなければなりません。

合致するデータベースが見つからない場合、`Open datastore` は Null を返します。

`localID` 引数は、リモートデータストア上で開かれるセッションのローカルエイリアスです。`localID` 引数の ID がすでにアプリケーションに存在している場合、その ID が使用されています。そうでない場合、データストアオブジェクトが使用されたときに `localID` のセッションが新規に作成されます。

`cs.Datastore` が提供するオブジェクトは、[ORDAマッピングルール](#) に基づいて、ターゲットデータベースからマッピングされます。

一旦セッションが開かれると、以下の 2行の宣言は同等のものとなり、同じデータストアオブジェクトへの参照を返します:

```
$myds:=Open datastore(connectionInfo;"myLocalId")
$myds2:=ds("myLocalId")
// $myds と $myds2 は同一のものです
```

`connectionInfo` には、接続したいリモートデータストアの詳細を格納したオブジェクトを渡します。オブジェクトは以下のプロパティを格納することができます (`hostname` を除き、すべてのプロパティは任意です):

プロパティ	タイプ	説明
hostname	Text	リモートデータストアの名前または IPアドレス + ":" + ポート番号 (ポート番号は必須)
user	Text	ユーザー名
password	Text	ユーザーパスワード
idleTimeout	Longint	アクティビティがなかった場合に、セッションがタイムアウトするまでの時間 (分単位)。この時間を過ぎると、4Dによって自動的にセッションが閉じられます。省略時のデフォルトは 60 (1時間) です。60 (分) 未満の値を指定することはできません (60 未満の値を渡した場合、タイムアウトは 60 (分) に設定されます)。詳細については、 セッションの終了 を参照ください。
tls	Boolean	安全な接続を使用します(*)。省略時のデフォルトは false です。可能なかぎり安全な接続を使用することが推奨されます。
type	Text	"4D Server" でなければなりません

(*) `tls` が true だった場合、以下の条件が満たされていれば、HTTPSプロトコルが使用されます:

- リモートデータストアで HTTPS が有効化されている。
- 指定されたポート番号は、データベース設定で設定されている HTTPS ポートと合致している。
- データベースに有効な証明書と非公開暗号鍵がインストールされている。条件を満たさない場合、エラー "1610 - ホスト xxx へのリモートリクエストに失敗しました" が生成されます。

例題 1

`user / password` を指定せずにリモートデータストアに接続します:

```

var $connectTo : Object
var $remoteDS : cs.DataStore
$connectTo:=New object("type";"4D Server";"hostname";"192.168.18.11:8044")
$remoteDS:=Open datastore($connectTo;"students")
ALERT("このリモートデータストアには "+String($remoteDS.Students.all().length)+" 名の生徒が登録されています")

```

例題 2

user / password / timeout / tls を指定してリモートデータストアに接続します:

```

var $connectTo : Object
var $remoteDS : cs.DataStore
$connectTo:=New object("type";"4D Server";"hostname";\"192.168.18.11:4443";\
    "user";"marie";"password";$pwd;"idleTimeout";70;"tls";True)
$remoteDS:=Open datastore($connectTo;"students")
ALERT("このリモートデータストアには "+String($remoteDS.Students.all().length)+" 名の生徒が登録されています")

```

例題 3

複数のリモートデータストアと接続します:

```

var $connectTo : Object
var $frenchStudents; $foreignStudents : cs.DataStore
$connectTo:=New object("hostname";"192.168.18.11:8044")
$frenchStudents:=Open datastore($connectTo;"french")
$connectTo.hostname:="192.168.18.11:8050"
$foreignStudents:=Open datastore($connectTo;"foreign")
ALERT("フランスの生徒は "+String($frenchStudents.Students.all().length)+" 名です")
ALERT("外国の生徒は "+String($foreignStudents.Students.all().length)+" 名です")

```

エラー管理

エラーが起きた場合、コマンドは Null を返します。リモートデータベースにアクセスできなかった場合（アドレス違い、Webサーバーが開始されていない、http/https が有効化されていない、等）、エラー1610 "ホスト XXX へのリモートリクエストに失敗しました" が生成されます。このエラーは ON ERR CALL で実装されたメソッドで割り込み可能です。

.dataclassName

▶ 履歴

.dataclassName : 4D.DataClass

説明

データストアの各データクラスは DataStore オブジェクト のプロパティとして利用可能です。戻り値のオブジェクトには データクラスの詳細が格納されています。

例題

```

var $emp : cs.Employee
var $sel : cs.EmployeeSelection
$emp:=ds.Employee // $emp は Employee データクラスを格納します
$sel:=$emp.all() // 全従業員のエンティティセレクションを取得します

// あるいは以下のように直接書くことも可能です:
$sel:=ds.Employee.all()

```

.cancelTransaction()

▶履歴

.cancelTransaction() | 引数 | タイプ | | 説明 | | -- | --- | ::| ----- | | | | このコマンドは引数を必要としません |

説明

.cancelTransaction() 関数は、指定データストアのカレントプロセスにおいて、.startTransaction() によって開かれた トランザクションをキャンセルします。

.cancelTransaction() 関数は、トランザクション中におこなわれたデータ変更をすべてキャンセルします。

複数のトランザクションをネストすること（サブトランザクション）が可能です。メイントランザクションがキャンセルされると、サブトランザクションも（たとえ個々に.validateTransaction() 関数で承認されていても）すべてキャンセルされます。

例題

.startTransaction() 関数の例題を参照ください。

.encryptionStatus()

▶履歴

.encryptionStatus(): Object

引数	タイプ		説明
戻り値	Object	<-	カレントデータストアと、各テーブルの暗号化についての情報

説明

.encryptionStatus() 関数は、カレントデータファイルの暗号化状態を示すオブジェクトを返します。カレントデータファイルとはつまり、ds データストアのデータファイルです。各テーブルの状態も提供されます。

その他のデータファイルの暗号化状態を調べるには、Data file encryption status コマンドを使います。

戻り値

戻り値のオブジェクトには、以下のプロパティが格納されています:

プロパティ			タイプ	説明
isEncrypted			Boolean	データファイルが暗号化されていれば true
keyProvided			Boolean	暗号化されたデータファイルに合致する暗号化キーが提供されていれば true (*)
tables			Object	暗号化可能および暗号化されたテーブルと同じ数のプロパティを持つオブジェクト
	tableName		Object	暗号化可能または暗号化されたテーブル
		name	Text	テーブル名
		num	Number	テーブル番号
		isEncryptable	Boolean	ストラクチャーファイルにおいて、テーブルが暗号化可能と宣言されれば true
		isEncrypted	Boolean	データファイルにおいて、テーブルのレコードが暗号化されていれば true

(*) 暗号化キーは、以下の手段のいずれかで提供されます:

- .provideDataKey() コマンド
- データストアを開く前に接続されていたデバイスのルート

- Discover data key コマンド

例題

カレントデータファイル内で暗号化されているテーブルの数を知りたい場合:

```

var $status : Object

$status:=dataStore.encryptionStatus()

If($status.isEncrypted) // データベースが暗号化されていれば
    C_LONGINT($vcount)
    C_TEXT($tabName)
    For each($tabName;$status.tables)
        If($status.tables[$tabName].isEncrypted)
            $vcount:=$vcount+1
        End if
    End for each
    ALERT("データベースには "+String($vcount)+" 件の暗号化されたテーブルが存在しています。")
Else
    ALERT("このデータベースは暗号化されていません。")
End if

```

.getInfo()

▶ 覆歴

.getInfo(): Object

引数	タイプ		説明
戻り値	Object	<-	データストアのプロパティ

説明

.getInfo() 関数は、データストアの情報を提供するオブジェクトを返します。このメソッドは汎用的なコードを書くのに有用です。

返されるオブジェクト

プロパティ	タイプ	説明															
type	string	<ul style="list-style-type: none"> "4D": ds で利用可能なメインデータストア "4D Server": Open datastore で開かれたリモートデータストア 															
networked	boolean	<ul style="list-style-type: none"> true: ネットワーク接続を介してアクセスされたデータストア false: ネットワーク接続を介さずにアクセスしているデータストア (ローカルデータベース) 															
localID	テキスト	マシン上のデータストアID。これは、Open datastore コマンドで返される localId 文字列です。メインデータストアの場合は空の文字列 ("") です。															
connection	object	リモートデータストア接続の情報を格納したオブジェクト (メインデータストアの場合は返されません)。次のプロパティを含みます: <table border="1"> <thead> <tr> <th>プロパティ</th> <th>タイプ</th> <th>説明</th> </tr> </thead> <tbody> <tr> <td>hostname</td> <td>テキスト</td> <td>リモートデータストアの IP アドレスまたは名称 + ":" + ポート番号</td> </tr> <tr> <td>tls</td> <td>boolean</td> <td>リモートデータストアとセキュア接続を利用している場合は true</td> </tr> <tr> <td>idleTimeout</td> <td>number</td> <td>セッション非アクティブタイムアウト (分単位)。</td> </tr> <tr> <td>user</td> <td>テキスト</td> <td>リモートデータストアにて認証されたユーザー</td> </tr> </tbody> </table>	プロパティ	タイプ	説明	hostname	テキスト	リモートデータストアの IP アドレスまたは名称 + ":" + ポート番号	tls	boolean	リモートデータストアとセキュア接続を利用している場合は true	idleTimeout	number	セッション非アクティブタイムアウト (分単位)。	user	テキスト	リモートデータストアにて認証されたユーザー
プロパティ	タイプ	説明															
hostname	テキスト	リモートデータストアの IP アドレスまたは名称 + ":" + ポート番号															
tls	boolean	リモートデータストアとセキュア接続を利用している場合は true															
idleTimeout	number	セッション非アクティブタイムアウト (分単位)。															
user	テキスト	リモートデータストアにて認証されたユーザー															

- `. getInfo()` 関数が、4D Server またはシングルユーザー版 4D 上で実行された場合、`networked` は `false` となります。
- `. getInfo()` 関数が、リモート版 4D 上で実行された場合、`networked` は `true` となります。

例題 1

```
var $info : Object

$info:=ds.getInfo() // 4D Server または 4D 上で実行した場合
//>{"type":"4D","networked":false,"localID":""}

$info:=ds.getInfo() // リモート版4D 上で実行した場合
//>{"type":"4D","networked":true,"localID":""}
```

例題 2

リモートデータストアの場合:

```
var $remoteDS : cs.DataStore
var $info; $connectTo : Object

$connectTo:=New object("hostname";"111.222.33.44:8044";"user";"marie";"password";"aaaa")
$remoteDS:=Open datastore($connectTo;"students")
$info:=$remoteDS.getInfo()

//>{"type":"4D Server",
// "localID":"students",
// "networked":true,
// "connection":{hostname:"111.222.33.44:8044",tls:false,idleTimeout:2880,user:"marie"}}
```

.getRequestLog()

▶ 覆歴

`.getRequestLog()` : Collection

引数	タイプ		説明
戻り値	Collection	<-	オブジェクトのコレクション (要素毎に一つのリクエストを記述します)

説明

`.getRequestLog()` 関数は、クライアント側のメモリに記録されている ORDAリクエストを返します。ORDAリクエストのログが、`.startRequestLog()` 関数によって事前に有効化されている必要があります。

このメソッド(リモートの 4D で呼び出す必要があり、そうでない場合には空のコレクションを返します。これはクライアント/サーバー環境でのデバッグを想定して設計されています)。

戻り値

スタックされたリクエストオブジェクトのコレクションが返されます。直近のリクエストにはインデックス 0 が振られています。

ORDAリクエストログのフォーマットの詳細は、[ORDAクライアントリクエスト](#) の章を参照ください。

例題

`.startRequestLog()` の例題2を参照ください。

.isAdminProtected()

▶ 覆歴

`.isAdminProtected() : Boolean`

引数	タイプ		説明
戻り値	Boolean	<-	データエクスプローラーへのアクセスが無効に設定されている場合は <code>true</code> 、有効の場合は <code>false</code> (デフォルト)

説明

`.isAdminProtected()` 関数は、現在のセッションにおいて [データエクスプローラー](#)へのアクセスが無効に設定されている場合は `true` を返します。

`webAdmin` セッションにおいて、データエクスプローラーへのアクセスはデフォルトで有効となっていますが、管理者によるデータアクセスを禁止するため無効にすることもできます ([.setAdminProtection\(\)](#) 関数参照)。

参照

[.setAdminProtection\(\)](#)

.makeSelectionsAlterable()

▶ 履歴

`.makeSelectionsAlterable() | 引数 | タイプ | | 説明 | | -- | --- | ::| ----- | | | | | このコマンドは引数を必要としません |`

説明

`.makeSelectionsAlterable()` 関数は、カレントアプリケーションのデータストアにおいて、すべての新規エンティティセレクションをデフォルトで追加可能に設定します ([リモートデータストア](#) を含む)。これはたとえば `On Startup` データベースメソッドなどで、一度だけ使用することが想定されています。

このメソッドが呼ばれてない場合、新規エンティティセレクションはそれぞれの "親" の性質や作成方法に応じて、共有可能に設定される場合もあります ([共有可能/追加可能なエンティティセレクション](#) 参照)。

この関数は、`OB Copy` または `.copy()` に `ck shared` オプションを明示的に使用して作成されたエンティティセレクションには適用されません。

互換性に関する注記: このメソッドは 4D v18 R5 より前のバージョンから変換されたプロジェクトで、`.add()` の呼び出しを使用しているものにおいてのみ使用してください。このコンテキストにおいては、`.makeSelectionsAlterable()` を使用することで、既存プロジェクト内で以前の 4D のふるまいを再現し、時間を節約できます。逆に、4D v18 R5 以降のバージョンで作成された新規プロジェクトにおいては、この関数の使用は 推奨されていません。エンティティセレクションを共有可能にできないため、パフォーマンスとスケーラビリティの観点で妨げになるからです。

.provideDataKey()

▶ 履歴

`.provideDataKey(curPassPhrase : Text) : Object`

`.provideDataKey(curDataKey : Object) : Object`

引数	タイプ		説明
curPassPhrase	Text	->	カレントのパスフレーズ
curDataKey	Object	->	カレントのデータ暗号化キー
戻り値	Object	<-	暗号化キーのチェックの結果

説明

`.provideDataKey()` 関数は、データストアのカレントデータファイルのデータ暗号化キーを受け取り、暗号化されたデータと合致するかどうかチェックします。この関数は、暗号化されたデータベースを開くときや、データファイルの再暗号化など暗号化キーが必要となる暗号化オペレーションを実行する際に使用します。

- `.provideDataKey()` 関数は暗号化されたデータベース内で呼び出される必要があります。暗号化されていないデータベース内で呼び出した場合、エラー-2003 (暗号化キーはデータと合致しません) が返されます。データベースが暗号化されているかどうかを調べるには `Data file encryption status` コマンドを使用します。
- リモートの 4D または暗号化されたリモートデータストアから、`.provideDataKey()` 関数を呼び出すことはできません。

`curPassPhrase` パラメーターを使用する場合は、データ暗号化キーの生成に使用した文字列を渡します。このパラメーターを使用した場合、暗号化キーが生成されます。

`curDataKey` パラメーターを使用する場合は、データ暗号化キー (`encodedKey` プロパティ) を格納するオブジェクトを渡します。このキーは、`New data key` コマンドで生成された可能性があります。

有効な暗号化キーが提供された場合、そのキーはメモリ内の `keyChain` に追加され、暗号化モードが有効になります:

- 暗号化可能テーブルに対するデータ編集はすべて、ディスク上 (.4DD, .journal、.4Dindx ファイル) で暗号化されます。
- 暗号化可能テーブルから読み出したすべてのデータは、メモリ内で復号化されます。

戻り値

コマンドの実行結果は、戻り値のオブジェクトに格納されます:

プロパティ		タイプ	説明
success		Boolean	提供された暗号化キーが暗号化データと合致すれば <code>true</code> 、それ以外は <code>false</code>
			以下のプロパティは、 <code>success</code> が <code>FALSE</code> であった場合にのみ返されます。
status		Number	エラーコード (提供された暗号化キーが間違っていた場合には 4)
statusText		Text	エラーメッセージ
errors		Collection	エラーのスタック。最初のエラーに最も高いインデックスが割り当てられます。
[].componentSignature		Text	内部コンポーネント名
[].errCode		Number	エラー番号
[].message		Text	エラーメッセージ

`curPassphrase` および `curDataKey` のどちらの引数も渡されなかった場合、`.provideDataKey()` は `null` を返します (この場合エラーは生成されません)。

例題

```

var $keyStatus : Object
var $passphrase : Text

$passphrase:=Request("パスフレーズを入力してください。")
If(OK=1)
    $keyStatus:=ds.provideDataKey($passphrase)
    If($keyStatus.success)
        ALERT("提供された暗号化キーは有効です。")
    Else
        ALERT("提供された暗号化キーは無効です。暗号化データの編集はできません。")
    End if
End if

```

.setAdminProtection()

▶ 補足

`.setAdminProtection(status : Boolean)`

引数	タイプ		説明
status	Boolean	->	webAdmin ポート上で、データエクスプローラーによるデータアクセスを無効にするには true、アクセスを有効にするには false (デフォルト)

説明

`.setAdminProtection()` 関数は、WebAdmin セッションにおける [データエクスプローラー](#) 含め、[Web管理ポート](#) 上でのデータアクセスを無効に設定することができます。

この関数が呼び出されなかった場合のデフォルトでは、データエクスプローラーを使用した WebAdmin 権限を持つセッションについて、Web管理ポート上のデータアクセスは常に許可されます。環境によっては（たとえば、アプリケーションサーバーが第三者のマシン上でホストされている場合）、管理者に対して [access key](#) 設定を含むサーバー設定の編集は許可しても、データ閲覧はできないようにしたいかもしれません。

このような場合にこの関数を呼び出すことで、ユーザー SESSION が WebAdmin 権限を持っていても、マシンの Web 管理ポート上のデータエクスプローラーによるデータアクセスを無効にすることができます。この関数を実行するとデータファイルは即座に保護され、そのステータスがディスク上に保存されます：アプリケーションを再起動しても、データファイルは保護されたままです。

例題

運用前に呼び出す `protectDataFile` プロジェクトメソッドを作成します：

```
ds.setAdminProtection(True) // データエクスプローラーによるデータアクセスを無効化します
```

参照

[.isAdminProtected\(\)](#)

.startRequestLog()

▶履歴

```
.startRequestLog()
.startRequestLog( file : 4D.File )
.startRequestLog( reqNum : Integer )
```

引数	タイプ		説明
file	4D.File	->	File オブジェクト
reqNum	Integer	->	メモリ内に保管するリクエストの数

説明

`.startRequestLog()` 関数は、クライアント側で ORDA リクエストのログを開始します。

このメソッドはリモート側の 4D で呼び出す必要があり、それ以外の場合には何もしません。これはクライアント/サーバー環境でのデバッグを想定して設計されています。

ORDA リクエストログは、渡した引数によってファイルまたはメモリに送ることができます：

- File コマンドで作成された file オブジェクトを渡した場合、ログデータはオブジェクト (JSON フォーマット) のコレクションとしてこのファイルに書き込まれます。各オブジェクトは一つのリクエストを表します。
ファイルがまだ存在しない場合には、作成されます。もしファイルが既に存在する場合、新しいログデータはそこに追加されていきます。メモリへのログ記録が既に始まっている状態で、`.startRequestLog()` が file 引数付きで呼び出された場合、メモリに記録されていたログは停止され消去されます。

JSON 評価を実行するには、ファイルの終わりに手動で] 文字を追加する必要があります。

- `reqNum` (倍長整数) 引数を渡した場合、メモリ内のログは（あれば）消去され、新しいログが初期化されます。`reqNum` 引数が指定する数にリクエスト数が到達するまでは、ログはメモリに保管され、到達した場合には古いエンタリーから消去されていきます (FIFO スタック)。

ファイルへのログ記録が既に始まっている状態で、`.startRequestLog()` が `reqNum` 引数付きで呼び出された場合、ファイルへのログは停止されます。

- 引数を何も渡さなかった場合、ログはメモリに記録されています。前もって `.startRequestLog()` が `reqNum` 引数付きで呼び出されていました場合（ただし `.stopRequestLog()` の前）、ログが次回消去されるかまたは `.stopRequestLog()` が呼び出されるまで、ログデータはメモリ内にスタックされます。

ORDAリクエストログのフォーマットの詳細は、[ORDAクライアントリクエスト](#) の章を参照ください。

例題 1

ORDA クライアントリクエストをファイルに記録し、ログシーケンス番号を使用します：

```
var $file : 4D.File
var $e : cs.PersonsEntity

$file:=File("/LOGS/ORDARequests.txt") // Logs フォルダー

SET DATABASE PARAMETER(Client Log Recording;1) // グローバルログシーケンス番号をトリガーします
ds.startRequestLog($file)
$e:=ds.Persons.get(30001) // リクエストを送信します
ds.stopRequestLog()
SET DATABASE PARAMETER(Client Log Recording;0)
```

例題 2

ORDA クライアントリクエストをメモリに記録します：

```
var $es : cs.PersonsSelection
var $log : Collection

ds.startRequestLog(3) // メモリにはリクエストを 3つまで保管します
$es:=ds.Persons.query("name=:1;"Marie")
$es:=ds.Persons.query("name IN :1";New collection("Marie"))
$es:=ds.Persons.query("name=:1;"So@")

$log:=ds.getRequestLog()
ALERT("The longest request lasted: "+String($log.max("duration"))+" ms")
```

.startTransaction()

▶ 履歴

`.startTransaction()` | 引数 | タイプ | 説明 | | -- | --- | | ----- | | | | このコマンドは引数を必要としません |

説明

`.startTransaction()` 関数は、対象データストアに対応するデータベース上で、カレントプロセス内のトランザクションを開始します。トランザクションプロセス中にデータストアのエンティティに加えられた変更は、トランザクションが確定されるかキャンセルされるまで一時的に保管されたままになります。

このメソッドがメインのデータストア（`ds` コマンドで返されるデータストア）で呼ばれた場合、トランザクションはメインのデータストアとそのデータベースで実行されるすべてのオペレーションに適用されます。これには、そこで実行される ORDA とクラシック言語も含まれます。

複数のトランザクションをネストすること（サブトランザクション）が可能です。個々のトランザクションまたはサブトランザクションは、それぞれキャンセルするか確定される必要があります。メイントランザクションがキャンセルされると、サブトランザクションも（たとえ個々に `.validateTransaction()` 関数で承認されていても）すべてキャンセルされます。

例題

```

var $connect; $status : Object
var $person : cs.PersonsEntity
var $ds : cs.DataStore
var $choice : Text
var $error : Boolean

Case of
  :($choice="local")
    $ds:=ds
  :($choice="remote")
    $connect:=New object("hostname";"111.222.3.4:8044")
    $ds:=Open datastore($connect;"myRemoteDS")
End case

$ds.startTransaction()
$person:=$ds.Persons.query("lastname=:1";"Peters").first()

If($person#Null)
  $person.lastname:="Smith"
  $status:=$person.save()
End if
...
...
If($error)
  $ds.cancelTransaction()
Else
  $ds.validateTransaction()
End if

```

.stopRequestLog()

▶ 補足

.stopRequestLog()
| 引数 | タイプ | | 説明 | | -- | --- | | ----- | | | | このコマンドは引数を必要としません |

説明

.stopRequestLog() 関数は、クライアント側の ORDAリクエストのログをすべて停止します (ファイル・メモリとも)。これは、開かれたドキュメントを実際に閉じてディスクに保存するため、ファイルにログを取っている場合に特に有用です。

このメソッドはリモート側の 4D で呼び出す必要があり、それ以外の場合には何もしません。これはクライアント/サーバー環境でのデバッグを想定して設計されています。

例題

[.startRequestLog\(\)](#) の例題を参照ください。

.validateTransaction()

▶ 補足

.validateTransaction()
| 引数 | タイプ | | 説明 | | -- | --- | | ----- | | | | このコマンドは引数を必要としません |

説明

.validateTransaction() 関数は、対象データストアの対応するレベルで [.startTransaction\(\)](#) で開始された トランザクションを受け入れます。

この関数は、トランザクション中におこなわれたデータストア上のデータの変更を保存します。

複数のトランザクションをネストすること (サブトランザクション) が可能です。メイントランザクションがキャンセルされると、サブトランザクションも (たとえ個々

(にこの関数で承認されても) すべてキャンセルされます。

例題

`.startTransaction()` の例題を参照ください。

Email

4Dにおけるメールの作成・送信・受信は `Email` オブジェクトの操作よっておこなわれます。

`transporter` クラス関数を使ってメールを取得する際に、`Email` オブジェクトが作成されます。

- IMAP - `.getMail()` および `.getMails()` 関数は IMAPサーバーからメールを受信します。
- POP3 - `.getMail()` 関数は POP3サーバーからメールを受信します。

また、`New object` 4Dコマンドを使って新規かつ空の `Email` オブジェクトを作成してから、`Email` オブジェクトプロパティを設定していくことも可能です。

`Email` オブジェクトは SMTP `.send()` 関数を使って送信します。

`MAIL Convert from MIME` および `MAIL Convert to MIME` コマンドは、MIME コンテンツから `Email` オブジェクトに、またはその逆の変換をおこなうのに使用できます。

Email オブジェクト

Email オブジェクトは次のプロパティを提供します:

4D は Email オブジェクトのフォーマットは [JMAP specification](#) に準拠します。

`.attachments : Collection`

`4D.MailAttachment` オブジェクトのコレクション

`.bcc : Text`

`.bcc : Object`

`.bcc : Collection`

非表示 (BCC: Blind Carbon Copy) のメール受信者 アドレス

`.bodyStructure : Object`

(任意) メッセージ本文の完全なMIME ストラクチャーである `EmailBodyPart` オブジェクト

`.bodyValues : Object`

(任意) `bodyStructure` の <partID> 毎にオブジェクトを格納している `EmailBodyValue` オブジェクト

`.cc : Text`

`.cc : Object`

`.cc : Collection`

追加 (CC: Carbon Copy) のメール受信者 アドレス

`.comments : Text`

追加のコメントのヘッダー

`.from : Text`

`.from : Object`

`.from : Collection`

メールの送信元 アドレス

`.headers : Collection`

メッセージ内で現れる順番どおりの `EmailHeader` オブジェクトのコレクション

`.htmlBody : Text`

(任意、SMTPのみ) HTML形式のメールメッセージ (デフォルトの文字セットは UTF-8)	
.id : Text	IMAP サーバーからの固有ID
.inReplyTo : Text	カレントメッセージが返信している、元のメッセージのメッセージID
.keywords : Object	各プロパティ名がキーワードであり、各値が true であるキーワードセットのオブジェクト
.messageId : Text	メッセージ識別ヘッダー ("message-id")
.receivedAt : Text	IMAPサーバーにメールが到着した時間の、ISO 8601 UTC フォーマットでのタイムスタンプ (例: 2020-09-13T16:11:53Z)
.references : Collection	返信チェーン内メッセージの、全メッセージID のコレクション
.replyTo : Text	
.replyTo : Object	
.replyTo : Collection	
	返信用 アドレス
.sendAt : Text	メールのタイムスタンプ (ISO 8601 UTCフォーマット)
.sender : Text	
.sender : Object	
.sender : Collection	
	メールのソース アドレス
.size : Integer	IMAPサーバーから返された Email オブジェクトのサイズ (バイト単位)
.subject : Text	メールの件名
.textBody : Text	(任意、SMTPのみ) 標準テキスト形式のメールメッセージ (デフォルトの文字セットは UTF-8)
.to : Text	
.to : Object	
.to : Collection	
	メールのメインの受信者 アドレス

メールアドレス

メールアドレスを格納するプロパティ (`from`, `cc`, `bcc`, `to`, `sender`, `replyTo`) はすべて、テキスト・オブジェクト・コレクション型の値を受け付けます。

テキスト

- 単一のメールアドレス: "somebody@domain.com"
- 単一の表示名+メールアドレス: "Somebody somebody@domain.com"
- 複数のメールアドレス: "Somebody somebody@domain.com, me@home.org"

オブジェクト

2つのプロパティを持つオブジェクト:

プロパティ	タイプ	説明
name	Text	表示名 (null も可能)
email	Text	メールアドレス

コレクション

アドレスオブジェクトのコレクション

メール本文の扱い

`textBody` および `htmlBody` はどちらも `SMTP.send()` でのみ使用され、これによって単純なメールの送信が可能になります。プロパティが両方ともある場合、MIME content-type の multipart/alternative が使用されます。メールクライアントは multipart/alternative パートを認識し、必要に応じてテキスト部または html 部を表示します。

`Email オブジェクト` が MIME ドキュメントからビルトされた場合 (例: `MAIL Convert from MIME` コマンドで生成されたとき) は、`bodyStructure` および `bodyValues` が `SMTP` に使用されます。この場合、`bodyStructure` および `bodyValues` プロパティは両方一緒に渡される必要があり、`textBody` および `htmlBody` の使用は推奨されません。

`bodyStructure` および `bodyValues` オブジェクトの例

```
"bodyStructure": {
    "type": "multipart/mixed",
    "subParts": [
        {
            "partId": "p0001",
            "type": "text/plain"
        },
        {
            "partId": "p0002",
            "type": "text/html"
        }
    ],
    "bodyValues": {
        "p0001": {
            "value": "I have the most brilliant plan. Let me tell you all about it."
        },
        "p0002": {
            "value": "<!DOCTYPE html><html><head><title></title><style type=\"text/css\">div{font-size:16px}</style><div>The most brilliant plan ever!</div></body></html>"
        }
    }
}
```

.attachments

`.attachments` : Collection

説明

`.attachments` プロパティは、`4D.MailAttachment` オブジェクトのコレクションを格納します。

`MailAttachment` オブジェクトは `MAIL New attachment` コマンドによって定義されます。`MailAttachment` オブジェクトは特有の [プロパティ](#) や [関数](#) を持ちます。

.bcc

.bcc : Text
.bcc : Object
.bcc : Collection

説明

.bcc プロパティは、非表示 (BCC: Blind Carbon Copy) のメール受信者 アドレスを格納します。

.bodyStructure

.bodyStructure : Object

説明

.bodyStructure プロパティは、(任意) メッセージ本文の完全なMIME ストラクチャーである *EmailBodyPart* オブジェクトを格納します。メール本文の扱い を参照ください。

.bodyStructure オブジェクトには、次のプロパティが格納されています:

プロパティ	タイプ	値
partID	Text	メールのパートを固有に識別する ID
type	Text	(必須) パートの Content-Type ヘッダーフィールドの値
charset	Text	Content-Type ヘッダーフィールドの Charset の値
encoding	Text	<code>isEncodingProblem=true</code> の場合、Content-Transfer-Encoding の値が追加されます (デフォルトでは未定義)
disposition	Text	パートの Content-Disposition ヘッダーフィールドの値
language	Text の Collection	パートの Content-Language ヘッダーフィールドの、RFC3282 で定義されている言語タグの一覧 (あれば)
location	Text	パートの Content-Location ヘッダーフィールドの、RFC2557 で定義されている URI (あれば)
subParts	Object の Collection	それぞれの子の本文パート (<i>EmailBodyPart</i> オブジェクトのコレクション)
headers	Object の Collection	パート内の全ヘッダーフィールドの、メッセージ内で出現する順の一覧 (<i>EmailHeader</i> オブジェクトのコレクション。headers プロパティ参照)

.bodyValues

.bodyValues : Object

説明

.bodyValues プロパティは、(任意) `bodyStructure` の `<partID>` 每にオブジェクトを格納している *EmailBodyValue* オブジェクトを格納します。メール本文の扱い を参照ください。

.bodyValues オブジェクトには、次のプロパティが格納されています:

プロパティ	タイプ	値
<code>partID.value</code>	テキスト	本文パートの値
<code>partID.isEncodingProblem</code>	boolean	文字セットをデコーディング中に、不正なフォーマットのセクション、未知の文字セット、あるいは未知の content-transfer-encoding が見つかった場合には true。デフォルトは false。

.cc

.cc : Text
.cc : Object
.cc : Collection

説明

.cc プロパティは、追加 (CC: Carbon Copy) のメール受信者 アドレスを格納します。

.comments

.comments : Text

説明

.comments プロパティは、追加のコメントのヘッダーを格納します。

コメントはメッセージのヘッダーセクション内にのみ表示されます (つまり本文部分には触れないということです)。

特定のフォーマット条件についての詳細は、[RFC#5322](#) を参照ください。

.from

.from : Text
.from : Object
.from : Collection

説明

.from プロパティは、メールの送信元 アドレスを格納します。

送信されるメールには、それぞれ sender および from アドレスの両方がついています:

- sender ドメインは、受信側のメールサーバーがセッションを開いたときに受け取るドメインです。
- from アドレスは、受信者から見えるアドレスです。

混乱を避けるため、sender および from アドレスには同じアドレスを使用することが推奨されます。

.headers

.headers : Collection

説明

.headers プロパティは、メッセージ内で現れる順番どおりの EmailHeader オブジェクトのコレクションを格納します。これによってユーザーは拡張された (登録された) ヘッダーや、ユーザー定義された (登録されていない、"X" で始まる) ヘッダーを追加することができます。

メールレベルでプロパティとして設定されている "from" または "cc" などのヘッダーを EmailHeader オブジェクトプロパティが定義している場合、EmailHeader プロパティは無視されます。

ヘッダーコレクションの各オブジェクトには、次のプロパティが格納されることがあります:

プロパティ	タイプ	値
[].name	テキスト	(必須) RFC#5322 で定義されているヘッダーフィールド名。null または未定義の場合には、ヘッダーフィールドは MIME ヘッダーに追加されません。
[].value	テキスト	RFC#5322 で定義されているヘッダーフィールド値。

.htmlBody

.htmlBody : Text

説明

.htmlBody プロパティは、(任意、SMTPのみ) HTML形式のメールメッセージ (デフォルトの文字セットは UTF-8)を格納します。 [メール本文の扱い](#) を参照ください。

.id

.id : Text

説明

[IMAP transporter](#) のみ。

.id プロパティは、IMAP サーバーからの固有IDを格納します。

.inReplyTo

.inReplyTo : Text

説明

.inReplyTo プロパティは、カレントメッセージが返信している、元のメッセージのメッセージIDを格納します。

特定のフォーマット条件についての詳細は、[RFC#5322](#) を参照ください。

.keywords

.keywords : Object

説明

.keywords プロパティは、各プロパティ名がキーワードであり、各値が true であるキーワードセットのオブジェクトを格納します。

このプロパティは "keywords" ヘッダーです ([RFC#4021](#) 参照)。

プロパティ	タイプ	値
.<keyword>	boolean	設定するキーワード (値は true でなければなりません)。

予約されたキーワード:

- \$draft - メッセージが下書きであることを表します
- \$seen - メッセージが読まれたことを表します
- \$flagged - メッセージが注視されるべきであることを表します (例: 至急のメール)
- \$answered - メッセージに返信がされたことを表します
- \$deleted - メッセージが消去されることを表します

例題

```
$mail.keywords["$flagged"] := True  
$mail.keywords["4d"] := True
```

.messageId

.messageId : Text

説明

.messageId プロパティは、メッセージ識別ヘッダー ("message-id")を格納します。

通常は、"lettersOrNumbers@domainname" の形式、たとえば "abcdef.123456@4d.com" などです。この固有ID は特にフォーラムや公開メーリングリストで使用されています。一般的に、メールサーバーは送信するメッセージにこのヘッダーを自動的に追加します。

.receivedAt

.receivedAt : Text

説明

[IMAP transporter](#) のみ。

.receivedAt プロパティは、IMAPサーバーにメールが到着した時間の、ISO 8601 UTC フォーマットでのタイムスタンプ (例: 2020-09-13T16:11:53Z)を格納します。

.references

.references : Collection

説明

.references プロパティは、返信チェーン内メッセージの、全メッセージID のコレクションを格納します。

特定のフォーマット条件についての詳細は、[RFC#5322](#) を参照ください。

.replyTo

.replyTo : Text

.replyTo : Object

.replyTo : Collection

説明

.replyTo プロパティは、返信用 [アドレス](#)を格納します。

.sendAt

.sendAt : Text

説明

.sendAt プロパティは、メールのタイムスタンプ (ISO 8601 UTCフォーマット)を格納します。

.sender

.sender : Text
.sender : Object
.sender : Collection

説明

.sender プロパティは、メールのソース [アドレス](#)を格納します。

送信されるメールには、それぞれ sender および [from](#) アドレスの両方がついています：

- sender ドメインは、受信側のメールサーバーがセッションを開いたときに受け取るドメインです。
- from アドレスは、受信者から見えるアドレスです。

混乱を避けるため、sender および from アドレスには同じアドレスを使用することが推奨されます。

.size

.size : Integer

説明

[IMAP transporter](#) のみ。

.size プロパティは、IMAPサーバーから返された Email オブジェクトのサイズ (バイト単位)を格納します。

.subject

.subject : Text

説明

.subject プロパティは、メールの件名を格納します。

.textBody

.textBody : Text

説明

.textBody プロパティは、(任意、SMTPのみ) 標準テキスト形式のメールメッセージ (デフォルトの文字セットは UTF-8)を格納します。 [メール本文の扱い](#) を参照ください。

.to

.to : Text
.to : Object
.to : Collection

説明

.to プロパティは、メールのメインの受信者 [アドレス](#)を格納します。

MAIL Convert from MIME

▶履歴

MAIL Convert from MIME(*mime* : Blob) : Object
MAIL Convert from MIME(*mime* : Text) : Object

引数	タイプ		説明
mime	Blob, Text	->	MIME形式のメール
戻り値	Object	<-	Email オブジェクト

説明

`MAIL Convert from MIME` コマンドは、MIMEドキュメントを有効な Emailオブジェクトへと変換します。

戻り値の Email オブジェクトのフォーマットは [JMAP specification](#) に準拠します。

`mime` には、変換する有効な MIME ドキュメントを渡します。これはどのメールサーバーまたはアプリケーションから提供されたものでも可能です。`mime` 引数として、BLOB またはテキストを渡すことができます。MIME がファイルから渡された場合、文字セットと改行コード変換に関する問題を避けるため、BLOB型の引数を使用することが推奨されます。

返されるオブジェクト

Email オブジェクト。

例題 1

テキストドキュメントとして保存された MIME のメールのテンプレートを読み込み、メールを送信します。

```
var $mime: Blob
var $mail;$server;$transporter;$status: Object

$mime:=File("/PACKAGE/Mails/templateMail.txt").getContent()

$mail:=MAIL Convert from MIME($mime)
$mail.to:="smith@mail.com"
$mail.subject:="Hello world"

$server:=New object
$server.host:="smtp.gmail.com"
$server.port:=465
$server.user:="test@gmail.com"
$server.password:="XXXX"

$transporter:=SMTP New transporter($server)
$status:=$transporter.send($mail)
```

例題 2

この例題では、ピクチャーが含まれた 4D Write Pro ドキュメントを直接送信します:

```

var $mime: Blob
var $email;$server;$transporter;$status: Object

// 4D Write Pro ドキュメントを MIME に書き出します
WP EXPORT VARIABLE(WParea;$mime;wk mime html)

// 4D Write Pro MIME 変数をメールオブジェクトに変換します
$email:=MAIL Convert from MIME($mime)

// Email オブジェクトのヘッダーを設定します
$email.subject:="4D Write Pro HTML body"
$email.from:="YourEmail@gmail.com"
$email.to:="RecipientEmail@mail.com"

$server:=New object
$server.host:="smtp.gmail.com"
$server.port:=465
$server.user:="YourEmail@gmail.com"
$server.password:="XXXX"

$transporter:=SMTP New transporter($server)
$status:=$transporter.send($email)

```

MAIL Convert to MIME

▶履歴

MAIL Convert to MIME(*mail* : Object { ; *options* : Object }) : Text

引数	タイプ		説明
mail	Object	->	Email オブジェクト
options	Object	->	文字セットとエンコーディングのメールオプション
戻り値	Text	<-	MIME に変換された Email オブジェクト

説明

MAIL Convert to MIME コマンドは、Email オブジェクトを MIME テキストへと変換します。このコマンドは、Email オブジェクトを送信する前に整形する目的で [SMTP_transporter.send\(\)](#) コマンドによって内部的に呼び出されます。また、オブジェクトの MIME フォーマットを解析するためにも使用されます。

mail には、変換するメールのコンテンツとストラクチャーの詳細を渡します。この情報には、メールアドレス（送信者と受信者）、メッセージそのもの、メッセージの表示タイプなどが含まれます。

Email オブジェクトのフォーマットは [JMAP specification](#) に準拠します。

options 引数を渡すと、メールに対して特定の文字セットとエンコーディング設定を指定することができます。次のプロパティを利用することができます：

プロパティ	タイプ	説明															
headerCharset	Text	<p>メールの以下の部分で使用される文字セットとエンコーディング: 件名、添付ファイル名、メール名の属性。とりうる値:</p> <table border="1"> <thead> <tr> <th>定数</th><th>値</th><th>説明</th></tr> </thead> <tbody> <tr> <td>mail mode ISO2022JP</td><td>US-ASCII_ISO-2022-JP_UTF8_QP</td><td> <ul style="list-style-type: none"> <i>headerCharset</i>: 可能なら US-ASCII、次に可能なら Japanese (ISO-2022-JP) & Quoted-printable、それも不可なら UTF-8 & Quoted-printable <i>bodyCharset</i>: 可能なら US-ASCII、次に可能なら Japanese (ISO-2022-JP) & 7-bit、それも不可なら UTF-8 & Quoted-printable </td></tr> <tr> <td>mail mode ISO88591</td><td>ISO-8859-1</td><td> <ul style="list-style-type: none"> <i>headerCharset</i>: ISO-8859-1 & Quoted-printable <i>bodyCharset</i>: ISO-8859-1 & 8-bit </td></tr> <tr> <td>mail mode UTF8</td><td>US-ASCII_UTF8_QP</td><td><i>headerCharset</i> & <i>bodyCharset</i>: 可能なら US-ASCII、それが不可なら UTF-8 & Quoted-printable (デフォルト値)</td></tr> <tr> <td>mail mode UTF8 in base64</td><td>US-ASCII_UTF8_B64</td><td><i>headerCharset</i> & <i>bodyCharset</i>: 可能な場合は US-ASCII、それ以外は UTF-8 & base64</td></tr> </tbody> </table>	定数	値	説明	mail mode ISO2022JP	US-ASCII_ISO-2022-JP_UTF8_QP	<ul style="list-style-type: none"> <i>headerCharset</i>: 可能なら US-ASCII、次に可能なら Japanese (ISO-2022-JP) & Quoted-printable、それも不可なら UTF-8 & Quoted-printable <i>bodyCharset</i>: 可能なら US-ASCII、次に可能なら Japanese (ISO-2022-JP) & 7-bit、それも不可なら UTF-8 & Quoted-printable 	mail mode ISO88591	ISO-8859-1	<ul style="list-style-type: none"> <i>headerCharset</i>: ISO-8859-1 & Quoted-printable <i>bodyCharset</i>: ISO-8859-1 & 8-bit 	mail mode UTF8	US-ASCII_UTF8_QP	<i>headerCharset</i> & <i>bodyCharset</i> : 可能なら US-ASCII、それが不可なら UTF-8 & Quoted-printable (デフォルト値)	mail mode UTF8 in base64	US-ASCII_UTF8_B64	<i>headerCharset</i> & <i>bodyCharset</i> : 可能な場合は US-ASCII、それ以外は UTF-8 & base64
定数	値	説明															
mail mode ISO2022JP	US-ASCII_ISO-2022-JP_UTF8_QP	<ul style="list-style-type: none"> <i>headerCharset</i>: 可能なら US-ASCII、次に可能なら Japanese (ISO-2022-JP) & Quoted-printable、それも不可なら UTF-8 & Quoted-printable <i>bodyCharset</i>: 可能なら US-ASCII、次に可能なら Japanese (ISO-2022-JP) & 7-bit、それも不可なら UTF-8 & Quoted-printable 															
mail mode ISO88591	ISO-8859-1	<ul style="list-style-type: none"> <i>headerCharset</i>: ISO-8859-1 & Quoted-printable <i>bodyCharset</i>: ISO-8859-1 & 8-bit 															
mail mode UTF8	US-ASCII_UTF8_QP	<i>headerCharset</i> & <i>bodyCharset</i> : 可能なら US-ASCII、それが不可なら UTF-8 & Quoted-printable (デフォルト値)															
mail mode UTF8 in base64	US-ASCII_UTF8_B64	<i>headerCharset</i> & <i>bodyCharset</i> : 可能な場合は US-ASCII、それ以外は UTF-8 & base64															
bodyCharset	Text	メールの HTML およびテキスト本文コンテンツで使用される文字セットとエンコーディング。取りうる値: headerCharset と同じ(上記参照)															

options 引数が省略された場合、ヘッダーおよび本文においては mail mode UTF8 設定が使用されます。

例題

```
var $mail: Object
var $mime: Text
$mail:=New object

// メール作成
$mail.from:="tsales@massmarket.com"
$mail.subject:="Terrific Sale! This week only!"
$mail.textBody:="Text format email"
$mail.htmlBody:="<html><body>HTML format email</body></html>"
$mail.to:=New collection
$mail.to.push(New object ("email";"noreply@4d.com"))
$mail.to.push(New object ("email";"test@4d.com"))

// Email オブジェクトを MIME に変換します
$mime:=MAIL Convert to MIME($mail)

// $mime の中身:
// MIME-Version: 1.0
// Date: Thu, 11 Oct 2018 15:42:25 GMT
// Message-ID: <7CA5D25B2B5E0047A36F2E8CB30362E2>
// Sender: tsales@massmarket.com
// From: tsales@massmarket.com
// To: noreply@4d.com
// To: test@4d.com
// Content-Type: multipart/alternative; boundary="E0AE5773D5E95245BBBBD80DD0687E218"
// Subject: Terrific Sale! This week only!
//
// --E0AE5773D5E95245BBBBD80DD0687E218
// Content-Type: text/plain; charset="UTF-8"
// Content-Transfer-Encoding: quoted-printable
//
// Text format email
// --E0AE5773D5E95245BBBBD80DD0687E218
// Content-Type: text/html; charset="UTF-8"
// Content-Transfer-Encoding: quoted-printable
//
// <html><body>HTML format email</body></html>
// --E0AE5773D5E95245BBBBD80DD0687E218--
```

Entity

レコードとテーブルの関係と同様に、エンティティはデータクラスのインスタンスです。エンティティはデータクラスと同じ属性を持つほか、データ値や、特有のプロパティおよび関数を持ちます。

概要

.attributeName : any

エンティティの属性値を格納します

.clone() : 4D.Entity

対象エンティティと同じレコードを参照する新規エンティティをメモリ内に作成します

.diff(entityToCompare : 4D.Entity { ; attributesToCompare : Collection }) : Collection

二つのエンティティの中身を比較し、その差異を返します

.drop({ mode : Integer }) : Object

データストアのエンティティに格納されているデータを削除します

.first() : 4D.Entity

対象エンティティが所属するエンティティセレクションの先頭エンティティへの参照を返します

.fromObject(filler : Object)

filler に指定した内容でエンティティの属性値を設定します

.getDataClass() : 4D.DataClass

エンティティのデータクラスを返します

.getKey({ mode : Integer }) : Text

.getKey({ mode : Integer }) : Integer

エンティティのプライマリーキー値を返します

.getRemoteContextAttributes() : Text

エンティティによって使われている最適化コンテキストの情報を返します

.getSelection() : 4D.EntitySelection

エンティティが所属するエンティティセレクションを返します

.getStamp() : Integer

エンティティのstampの値を返します

.indexOf({ entitySelection : 4D.EntitySelection }) : Integer

エンティティセレクション内におけるエンティティの位置を返します

.isNew() : Boolean

対象エンティティが作成されたばかりで、まだデータストアに保存されていない場合に true を返します

.last() : 4D.Entity

対象エンティティが所属するエンティティセレクションの最終エンティティへの参照を返します

.lock({ mode : Integer }) : Object

対象エンティティが参照するレコードにペシミスティック・ロックをかけます

.next() : 4D.Entity

エンティティが所属するエンティティセレクションの次のエンティティへの参照を返します

.previous() : 4D.Entity

エンティティが所属するエンティティセレクションの前のエンティティへの参照を返します

.reload() : Object

エンティティの中身をメモリ内にリロードします

.save({ mode : Integer }) : Object

エンティティの変更内容を保存します

.toObject() : Object

.toObject(filterString : Text { ; options : Integer }) : Object

.toObject(filterCol : Collection { ; options : Integer }) : Object

エンティティからビルトされたオブジェクトを返します

.touched() : Boolean

エンティティがメモリに読み込まれてから、あるいは保存されてから、エンティティ属性が変更されたかどうかをテストします

.touchedAttributes() : Collection

メモリに読み込み後に変更されたエンティティの属性名を返します

.unlock() : Object

対象エンティティが参照するレコードのペシミスティック・ロックを解除します

.attributeName

▶履歴

.attributeName : any

説明

データクラス属性はすべてエンティティのプロパティとして利用可能です。各エンティティのプロパティは、当該 エンティティの属性値を格納します。

データクラス属性は [] を使用したシンタックスを使用することでもアクセス可能です。

この属性値タイプは属性の種類 (.kind; リレーションまたはストレージ) によります。

- .attributeName で指定した属性がストレージ型の場合: .attributeName は attributeName と同じ型の値を返します。
- .attributeName で指定した属性がリレートエンティティ型の場合: .attributeName はリレートエンティティを返します。リレートエンティティの値は、ドット記法でプロパティを繋げることでアクセス可能です。例: "myEntity.employer.employees[0].lastname"
- .attributeName で指定した属性がリレートエンティティズ型の場合: .attributeName はリレートエンティティの新しいエンティティセレクションを返します。重複しているエンティティは取り除かれます (返されるのは順列なしのエンティティセレクションです)。

例題

```
var $myEntity : cs.EmployeeEntity
$myEntity:=ds.Employee.new() // エンティティを新規作成します
$myEntity.name:="Dupont" // 'Dupont' を 'name' 属性に代入します
$myEntity.firstname:="John" // 'John' を 'firstname' 属性に代入します
$myEntity.save() // エンティティを保存します
```

.clone()

▶履歴

.clone() : 4D.Entity

引数	タイプ		説明
戻り値	4D.Entity	<-	同レコードを参照する新しいエンティティ

説明

.clone() 関数は、対象エンティティと同じレコードを参照する新規エンティティをメモリ内に作成します。このメソッドを使用するとエンティティを個別に更新することができます。

エンティティに対して何らかの変更をおこなった場合、それらは `.save()` 関数が実行されたときのみ、参照先のレコードに保存されるという点に注意してください。

この関数は、すでにデータベースに保存されているエンティティに対してのみ使用可能です。新規に作成されたエンティティ(`.isNew()` が true を返すもの)に対して呼び出すことはできません。

例題

```
var $emp; $empCloned : cs.EmployeeEntity
$emp:=ds.Employee.get(672)
$empCloned:=$emp.clone()

$emp.lastName:="Smith" // $emp に対する変更は $empCloned には適用されません
```

.diff()

▶ 補足

.diff(*entityToCompare* : 4D.Entity { ; *attributesToCompare* : Collection }) : Collection

引数	タイプ		説明
<i>entityToCompare</i>	4D.Entity	->	対象エンティティと比較するエンティティ
<i>attributesToCompare</i>	Collection	->	比較する属性の名称
戻り値	Collection	<-	エンティティ間の差異

説明

.diff() 関数は、二つのエンティティの中身を比較し、その差異を返します。

entityToCompare には、オリジナルのエンティティと比較をするエンティティを渡します。

attributesToCompare 引数で、比較する属性を指定することができます。これを渡した場合、指定された属性に対してのみ比較がおこなわれます。省略時には、エンティティ間の差異がすべて返されます。

エンティティの差異は、以下のプロパティを持つオブジェクトのコレクションとして返されます:

プロパティ名	タイプ	説明
<i>attributeName</i>	String	属性名
<i>value</i>	any - 属性の型による	オリジナルエンティティの属性値
<i>otherValue</i>	any - 属性の型による	<i>entityToCompare</i> の属性値

コレクションに含まれるのは異なる値を持っていた属性のみです。差異が見つからない場合、`diff()` は空のコレクションを返します。

この関数は、種類 (`kind`) が storage あるいは relatedEntity であるプロパティに適用されます。リレート先のエンティティそのものが変更された場合 (外部キーの変更)、リレーションの名称とそのプライマリーキー名が `attributeName` プロパティに返されます (リレーション名についての `value` および

`otherValue` は空になります)。

比較するどちらかのエンティティが Null である場合、エラーが生成されます。

例題 1

```
var $diff1; $diff2 : Collection
employee:=ds.Employee.query("ID=1001").first()
$clone:=employee.clone()
employee.firstName:="MARIE"
employee.lastName:="SOPHIE"
employee.salary:=500
$diff1:=$clone.diff(employee) // すべての差異が返されます
$diff2:=$clone.diff(employee;New collection"firstName";"lastName"))
// firstName と lastName についての差異のみが返されます
```

\$diff1:

```
[  
  {  
    "attributeName": "firstName",  
    "value": "Natasha",  
    "otherValue": "MARIE"  
  },  
  {  
    "attributeName": "lastName",  
    "value": "Locke",  
    "otherValue": "SOPHIE"  
  },  
  {  
    "attributeName": "salary",  
    "value": 66600,  
    "otherValue": 500  
  }  
]
```

\$diff2:

```
[  
  {  
    "attributeName": "firstName",  
    "value": "Natasha",  
    "otherValue": "MARIE"  
  },  
  {  
    "attributeName": "lastName",  
    "value": "Locke",  
    "otherValue": "SOPHIE"  
  }  
]
```

例題 2

```

var vCompareResult1; vCompareResult2; vCompareResult3; $attributesToInspect : Collection
vCompareResult1:=New collection
vCompareResult2:=New collection
vCompareResult3:=New collection
$attributesToInspect:=New collection

$e1:=ds.Employee.get(636)
$e2:=ds.Employee.get(636)

$e1.firstName:=$e1.firstName+" update"
$e1.lastName:=$e1.lastName+" update"

$c:=ds.Company.get(117)
$e1.employer:=$c
$e2.salary:=100

$attributesToInspect.push("firstName")
$attributesToInspect.push("lastName")

vCompareResult1:=$e1.diff($e2)
vCompareResult2:=$e1.diff($e2;$attributesToInspect)
vCompareResult3:=$e1.diff($e2;$e1.touchedAttributes())

```

vCompareResult1 (すべての差異が返されています):

```
[
{
  "attributeName": "firstName",
  "value": "Karla update",
  "otherValue": "Karla"
},
{
  "attributeName": "lastName",
  "value": "Marrero update",
  "otherValue": "Marrero"
},
{
  "attributeName": "salary",
  "value": 33500,
  "otherValue": 100
},
{
  "attributeName": "employerID", // プライマリーキー名
  "value": 117,
  "otherValue": 118
},
{
  "attributeName": "employer", // リレーション名
  "value": "[object Entity]",// Company のエンティティ 117
  "otherValue": "[object Entity]"// Company のエンティティ 118
}
]
```

vCompareResult2 (\$attributesToInspect についての差異のみ返されます)

```
[
  {
    "attributeName": "firstName",
    "value": "Karla update",
    "otherValue": "Karla"
  },
  {
    "attributeName": "lastName",
    "value": "Marrero update",
    "otherValue": "Marrero"
  }
]
```

vCompareResult3 (\$e1 において更新された (touch された) 属性のみが返されます)

```
[
  {
    "attributeName": "firstName",
    "value": "Karla update",
    "otherValue": "Karla"
  },
  {
    "attributeName": "lastName",
    "value": "Marrero update",
    "otherValue": "Marrero"
  },
  {
    "attributeName": "employerID", // プライマリーキー名
    "value": 117,
    "otherValue": 118
  },
  {
    "attributeName": "employer", // リレーション名
    "value": "[object Entity]", // Company のエンティティ 117
    "otherValue": "[object Entity]" // Company のエンティティ 118
  }
]
```

.drop()

▶履歴

.drop({mode : Integer}) : Object

引数	タイプ		説明
mode	Integer	->	dk force drop if stamp changed : スタンプが変更されていた場合でも強制的にドロップする
戻り値	Object	<-	ドロップの結果

説明

.drop() 関数は、データクラスに対応するテーブルにおいて、データストアのエンティティに格納されているデータを削除します。エンティティそのものはメモリ内に残るという点に注意してください。

マルチユーザー、あるいはマルチプロセスアプリケーションにおいて、.drop() 関数は "オプティミスティック・ロック" 機構のもとで実行されます。これはコードが保存されるたびに内部的なロックスタンプが自動的に増分していくという機構です。

mode 引数を渡さなかった場合のデフォルトでは、同エンティティが他のプロセスまたはユーザーによって変更されていた場合（つまり、スタンプが変更されていた場合）にエラーを返します（以下参照）。

`mode` に `dk force drop if stamp changed` オプションを渡すと、スタンプが変更されてもエンティティはドロップされます（プライマリーキーは変わらない場合）。

戻り値

`.drop()` によって返されるオブジェクトには以下のプロパティが格納されます：

プロパティ		タイプ	説明
success		boolean	ドロップが成功した場合には <code>true</code> 、それ以外は <code>false</code>
			エラーの場合にのみ利用可能：
status(*)		number	エラーコード、以下参照
statusText(*)		テキスト	エラーの詳細、以下参照
			ペシミスティック・ロックエラーの場合にのみ利用可能：
lockKindText		テキスト	"Locked by record"
lockInfo		object	ロック元についての情報
	task_id	number	プロセスID
	user_name	テキスト	マシン上でセッションユーザー名
	user4d_alias	テキスト	<code>SET USER ALIAS</code> で設定されていればユーザーイリアス。それ以外は 4D ディレクトリのユーザー名
	host_name	テキスト	マシン名
	task_name	テキスト	プロセス名
	client_version	テキスト	
			深刻なエラーの場合にのみ利用可能（深刻なエラーとは、プライマリーキーを重複させようとした、ディスクがいっぱいであった、などです）：
errors		Object の Collection	
	message	テキスト	エラーメッセージ
	component signature	テキスト	内部コンポーネント署名（例 "dmbg" はデータベースコンポーネントを表します）
	errCode	number	エラーコード

(*) エラー時には `Result` オブジェクトの `status` あるいは `statusText` プロパティに以下のいずれかの値が返されます：

定数	値	説明
dk status entity does not exist anymore	5	エンティティはもうデータ内に存在していません。このエラーは以下のような場合に起きます: <ul style="list-style-type: none"> エンティティがドロップされている（スタンプが変更されていて、メモリ空間は解放されている） エンティティがドロップされていて、他のプライマリーキー値を持つエンティティで置き換えられている（スタンプは変更されていて、新しいエンティティがメモリ空間を使用している）。entity.drop() を使用するとき、このエラーは dk force drop if stamp changed オプションを使用した場合に返されることがあります。entity.lock() を使用するとき、このエラーは dk reload drop if stamp changed オプションを使用した場合に返されることがあります。 割り当てられた statusText : "エンティティはもう存在しません"
dk status locked	3	エンティティはペシミスティック・ロックでロックされています。 割り当てられた statusText : "既にロックされています"
dk status serious error	4	深刻なエラーとは、低レベルのデータベースエラー（例：重複キー）、ハードウェアエラーなどです。 割り当てられた statusText : "その他のエラー"
dk status stamp has changed	2	エンティティの内部的なスタンプ値がデータ内に保存されているエンティティのものと合致しません（オptyimistic lock）。 <ul style="list-style-type: none"> entity.save() の場合: dk auto merge オプションが使用されていない場合に限りエラー entity.drop() の場合: dk force drop if stamp changed オプションが使用されていない場合に限りエラー entity.lock() の場合: dk reload if stamp changed オプションが使用されていない場合に限りエラー 割り当てられた statusText : "スタンプが変更されています"

例題 1

dk force drop if stamp changed オプションを使用しない例:

```

var $employees : cs.EmployeeSelection
var $employee : cs.EmployeeEntity
var $status : Object
$employees:=ds.Employee.query("lastName=:1";"Smith")
$employee:=$employees.first()
$status:=$employee.drop()
Case of
  :($status.success)
    ALERT($employee.firstName+" "+$employee.lastName+" をドロップしました。") // ドロップされたエンティティは削除されない
  :($status.status=dk status stamp has changed)
    ALERT($status.statusText)
End case

```

例題 2

dk force drop if stamp changed オプションを使用する例:

```

var $employees : cs.EmployeeSelection
var $employee : cs.EmployeeEntity
var $status : Object
$employees:=ds.Employee.query("lastName=:1";"Smith")
$employee:=$employees.first()
$status:=$employee.drop(dk force drop if stamp changed)
Case of
    :($status.success)
        ALERT($employee.firstName+" "+$employee.lastName+" をドロップしました。") // ドロップされたエンティティは
    :($status.status=dk status entity does not exist anymore)
        ALERT($status.statusText)
End case

```

.first()

▶ 覆歴

.first(): 4D.Entity

引数	タイプ		説明
戻り値	4D.Entity	<-	エンティティセレクションの先頭エンティティへの参照 (見つからなければ null)

説明

.first() 関数は、対象エンティティが所属するエンティティセレクションの先頭エンティティへの参照を返します。

対象エンティティが所属する既存エンティティセレクションが存在しない場合 (つまり `entity.getSelection()` が Null を返す場合)、関数は Null 値を返します。

例題

```

var $employees : cs.EmployeeSelection
var $employee; $firstEmployee : cs.EmployeeEntity
$employees:=ds.Employee.query("lastName = :1";"H@") // このエンティティセレクションは 3件のエンティティを持ちます
$employee:=$employees[2]
$firstEmployee:=$employee.first() // $firstEmployee は、$employees エンティティセレクションの先頭エンティティで

```

.fromObject()

▶ 覆歴

.fromObject(*filler* : Object)

引数	タイプ		説明
<i>filler</i>	Object	->	エンティティの属性値を設定するオブジェクト

説明

.fromObject() 関数は、*filler* に指定した内容でエンティティの属性値を設定します。

このコマンドは、元のエンティティを変更します。

オブジェクトとエンティティ間のマッピングは属性名でおこなわれます:

- オブジェクトのプロパティがデータクラスに存在しない場合、それは無視されます。
- データタイプは同じである必要があります。オブジェクトとデータクラス間で型が合致しない場合、4D は可能であればデータを変換しようとし ([データ](#))

[ターゲットの変換](#)) 参照)、それ以外の場合にはその属性は更新されません。

- プライマリーキーはそのまま、あるいは "__KEY" プロパティを (プライマリーキー値とともに) 使って指定することができます。その値のエンティティがデータクラス内に存在しない場合には、[.save\(\)](#) が呼び出されたときに指定値を使ってエンティティが作成されます。プライマリーキーを指定していない場合、エンティティは作成され、データベースのルールに基づいてプライマリーキー値が割り当てられます。自動インクリメント機能はプライマリーキーが null の場合にのみ計算されます。

filler 引数のオブジェクトは、以下の条件のいずれかを満たしている場合にはリレートエンティティを扱うことができます:

- *filler* が外部キーを格納している
- *filler* が、リレートエンティティ名と同じ名称のプロパティを格納しており、その値であるオブジェクトは "__KEY" という名称の単一のプロパティを格納している
- リレートエンティティが存在しない場合、無視されます。

例題

以下のような \$o オブジェクトがある場合:

```
{  
    "firstName": "Mary",  
    "lastName": "Smith",  
    "salary": 36500,  
    "birthDate": "1958-10-27T00:00:00.000Z",  
    "woman": true,  
    "managerID": 411, // リレートエンティティを主キー属性値で指定します  
    "employerID": 20 // リレートエンティティを主キー属性値で指定します  
}
```

以下のコードを実行すると、manager および employer というリレートエンティティを持つエンティティを作成します。

```
var $o : Object  
var $entity : cs.EmpEntity  
$entity:=ds.Emp.new()  
$entity.fromObject($o)  
$entity.save()
```

また、オブジェクトとして提供されたリレートエンティティを使用することもできます:

```
{  
    "firstName": "Marie",  
    "lastName": "Lechat",  
    "salary": 68400,  
    "birthDate": "1971-09-03T00:00:00.000Z",  
    "woman": false,  
    "employer": { // リレートエンティティをオブジェクトで指定します  
        "__KEY": "21"  
    },  
    "manager": { // リレートエンティティをオブジェクトで指定します  
        "__KEY": "411"  
    }  
}
```

.getDataClass()

▶ [履歴](#)

.getDataClass() : 4D.DataClass

引数	タイプ		説明
戻り値	4D.DataClass	<-	エンティティが所属している DataClass オブジェクト

説明

.getDataClass() 関数は、エンティティのデータクラスを返します。この関数は汎用的なコードを書くのに有用です。

例題

以下の汎用的なコードは、あらゆるエンティティを複製します：

```
// duplicate_entity メソッド
// duplicate_entity($entity)

#DECLARE($entity : 4D.Entity)
var $entityNew : 4D.Entity
var $status : Object

$entityNew:=$entity.getDataClass().new() // 親データクラスに新しいエンティティを作成します
$entityNew.fromObject($entity.toObject()) // 全属性を取得します
$entityNew[$entity.getDataClass().getInfo().primaryKey]:=Null // プライマリーキーをリセットします
$status:=$entityNew.save() // 複製したエンティティを保存します
```

.getKey()

▶ 履歴

.getKey({ mode : Integer }) : Text
.getKey({ mode : Integer }) : Integer

引数	タイプ		説明
mode	Integer	->	dk key as string : プライマリーキーの型にかかわらず、プライマリーキーを文字列として返します
戻り値	Text	<-	エンティティのテキスト型プライマリーキーの値
戻り値	Integer	<-	エンティティの数値型プライマリーキーの値

説明

.getKey() 関数は、エンティティのプライマリーキー値を返します。

プライマリーキーは数値 (倍長整数) あるいは文字列です。mode 引数として dk key as string オプションを渡すことで、実際のプライマリーキーの型に関係なく、返されるプライマリーキー値の型を文字列に "強制" することができます。

例題

```
var $employees : cs.EmployeeSelection
var $employee : cs.EmployeeEntity
$employees:=ds.Employee.query("lastName=:1";"Smith")
$employee:=$employees[0]
ALERT("プライマリーキー: "+$employee.getKey(dk key as string))
```

.getRemoteContextAttributes()

▶ 履歴

.getRemoteContextAttributes() : Text

引数	タイプ		説明
result	テキスト	<-	エンティティにリンクされたコンテキスト属性 (カンマ区切り)

上級者向け: この機能は、特定の構成のため、ORDAのデフォルト機能をカスタマイズする必要がある開発者向けです。ほとんどの場合、使用する必要はないでしょう。

説明

`.getRemoteContextAttributes()` 関数は、エンティティによって使われている最適化コンテキストの情報を返します。

エンティティについて [最適化コンテキスト](#) が存在しない場合、関数は空のテキストを返します。

例題

```

var $ds : 4D.DataStoreImplementation
var $address : cs.AddressEntity
var $p : cs.PersonsEntity
var $contextA : Object
var $info : Text
var $text : Text

$ds:=Open datastore(New object("hostname"; "www.myserver.com"); "myDS")

$contextA:=New object("context"; "contextA")

$address:=$ds.Address.get(1; $contextA)
$text:=""
For each ($p; $address.persons)
    $text:=$p.firstname+" "+$p.lastname
End for each

$info:=$address.getRemoteContextAttributes()

// $info = "persons, persons.lastname, persons.firstname"

```

参照

[EntitySelection.getRemoteContextAttributes\(\)](#)
[.clearAllRemoteContexts\(\)](#)
[.getRemoteContextInfo\(\)](#)
[.getAllRemoteContexts\(\)](#)
[.setRemoteContextInfo\(\)](#)

.getSelection()

▶ 覆歴

`.getSelection(): 4D.EntitySelection`

引数	タイプ		説明
戻り値	4D.EntitySelection	<-	エンティティが所属するエンティティセレクション (見つからなければ null)

説明

`.getSelection()` 関数は、エンティティが所属するエンティティセレクションを返します。

対象エンティティがエンティティセレクションに所属していない場合、関数は Null 値を返します。

例題

```

var $emp : cs.EmployeeEntity
var $employees; $employees2 : cs.EmployeeSelection
$emp:=ds.Employee.get(672) // エンティティセレクションに属していないエンティティです
$employees:=$emp.getSelection() // $employees は Null です

$employees2:=ds.Employee.query("lastName=:1";"Smith") // このエンティティセレクションは 6件のエンティティを格納し
$emp:=$employees2[0] // エンティティセレクションに所属しているエンティティです

ALERT("エンティティセレクションには "+String($emp.getSelection().length)+" 件のエンティティが含まれています")

```

.getStamp()

▶履歴

.getStamp() : Integer

引数	タイプ	説明
戻り値	整数	<- エンティティのスタンプ (エンティティが作成されたばかりの場合には 0)

説明

.getStamp() 関数は、エンティティのスタンプの値を返します。

内部スタンプは、エンティティが保存されるたびに 4D によって自動的にインクリメントされます。これは同じエンティティに対する複数のユーザーの同時アクセス・編集を管理します。この機構の詳細については、[エンティティロック](#) を参照ください。

(一度も保存されていない) 新規エンティティに対しては、このメソッドは 0 を返します。しかしながら、エンティティがまだ作成されたばかりかどうかを調べるには、[isNew\(\)](#) の使用が推奨されます。

例題

```

var $entity : cs.EmployeeEntity
var $stamp : Integer

$entity:=ds.Employee.new()
$entity.lastname:="Smith"
$entity.save()
$stamp:=$entity.getStamp() // $stamp=1

$entity.lastname:="Wesson"
$entity.save()
$stamp:=$entity.getStamp() // $stamp=2

```

.indexOf()

▶履歴

.indexOf({ entitySelection : 4D.EntitySelection }) : Integer

引数	タイプ		説明
entitySelection	4D.EntitySelection	->	エンティティの位置を取得する対象のエンティティセレクション
戻り値	整数	<-	エンティティセレクション内でのエンティティの位置

説明

.indexOf() 関数は、エンティティセレクション内におけるエンティティの位置を返します。

entitySelection 引数が渡されなかった場合はデフォルトで、所属エンティティセレクション内のエンティティの位置が返されます。 *entitySelection* 引数を渡した場合は、指定されたエンティティセレクション内のエンティティの位置を返します。

戻り値は、0 と、エンティティセレクションの length より 1 を引いた値の範囲内の数値です。

- エンティティがエンティティセレクションを持たない場合、あるいは *entitySelection* 引数で指定したエンティティセレクションに含まれていない場合には、-1 が返されます。
- entitySelection* 引数で指定したエンティティセレクションが Null である、あるいはエンティティと同じデータクラスのものでない場合には、エラーが生成されます。

例題

```
var $employees : cs.EmployeeSelection
var $employee : cs.EmployeeEntity
$employees:=ds.Employee.query("lastName = :1";"H@") // このエンティティセレクションには 3件のエンティティが格納さ
$employee:=$employees[1] // このエンティティはエンティティセレクションに所属しています
ALERT("The index of the entity in its own entity selection is "+String($employee.indexOf())) // 1

$employee:=ds.Employee.get(725) // エンティティセレクションに所属していないエンティティです
ALERT("The index of the entity is "+String($employee.indexOf())) // -1
```

.isNew()

▶履歴

.isNew() : Boolean

引数	タイプ		説明
戻り値	ブール	<-	エンティティが作成されたばかりで未保存の場合は true。それ以外は false。

説明

.isNew() 関数は、対象エンティティが作成されたばかりで、まだデータストアに保存されていない場合に true を返します。そうでない場合には、false を返します。

例題

```
var $emp : cs.EmployeeEntity
$emp:=ds.Employee.new()

If($emp.isNew())
    ALERT("新規エンティティです。")
End if
```

.last()

▶履歴

.last() : 4D.Entity

引数	タイプ		説明
戻り値	4D.Entity	<-	エンティティセレクションの最終エンティティへの参照（見つからなければ null）

説明

.last() 関数は、対象エンティティが所属するエンティティセレクションの最終エンティティへの参照を返します。

対象エンティティが所属する既存エンティティセレクションが存在しない場合（つまり `entity.getSelection()` が Null を返す場合）、関数は Null 値を返します。

例題

```
var $employees : cs.EmployeeSelection
var $employee; $lastEmployee : cs.EmployeeEntity
$employees:=ds.Employee.query("lastName = :1";"Hq") // このエンティティセレクションは 3件のエンティティを持ちます
$employee:=$employees[0]
$lastEmployee:=$employee.last() // $lastEmployee は、$employees エンティティセレクションの最終エンティティです
```

.lock()

▶履歴

`.lock({ mode : Integer }) : Object`

引数	タイプ		説明
mode	整数	->	<code>dk reload if stamp changed</code> : スタンプが変更される場合はロック前にリロードします
戻り値	オブジェクト	<-	ロックの結果

説明

`.lock()` 関数は、対象エンティティが参照するレコードにペシミスティック・ロックをかけます。ロックはレコードと、カレントプロセス内の当該エンティティの参照すべてに対してかけられます。

他のプロセスからはこのレコードがロックされて見えます（この関数を使って同エンティティをロックしようとした場合、`result.success` プロパティには `false` が返されます）。ロックをおこなったセッション内で実行される関数のみが、当該エンティティの属性を編集・保存できます。他のセッションは同エンティティを読み取り専用にロードできますが、値の入力・保存はできません。

`.lock()` でロックされたレコードは、以下の場合にロック解除されます：

- 同プロセス内で合致するエンティティに対して `.unlock()` 関数が呼び出された場合
- メモリ内のどのエンティティからも参照されなくなった場合、自動的にロックが解除されます。たとえば、エンティティのローカル参照に対してのみロックがかかっていた場合、関数の実行が終了すればロックは解除されます。メモリ内にエンティティへの参照がある限り、レコードはロックされたままです。

エンティティは REST セッションによってロックされる 場合もあります。

`mode` 引数を渡さなかった場合のデフォルトでは、同エンティティが他のプロセスまたはユーザーによって変更されていた場合（つまり、スタンプが変更されていた場合）にエラーを返します（以下参照）。

`mode` に `dk reload if stamp changed` オプションを渡すと、スタンプが変更されてもエラーは返されず、エンティティは再読み込みされます（エンティティが引き続き存在し、プライマリーキーも変わらない場合）。

戻り値

`.lock()` によって返されるオブジェクトには以下のプロパティが格納されます：

プロパティ		タイプ	説明
success		boolean	ロックに成功した場合 (あるいはエンティティがすでにカレントプロセスでロックされていた場合) には true、それ以外は false
			<code>dk reload if stamp changed</code> オプションが使用されていた場合にのみ利用可能:
wasReloaded		boolean	エンティティがリロードされ、かつリロードに成功した場合には true、それ以外は false
			エラーの場合にのみ利用可能:
status(*)		number	エラーコード、以下参照
statusText(*)		テキスト	エラーの詳細、以下参照
			ペシミスティック・ロックエラーの場合にのみ利用可能:
lockKindText		テキスト	"Locked by record" 4Dプロセスによるロック、"Locked by session" RESTセッションによるロック
lockInfo		object	ロック元についての情報。返されるプロパティはロック元 (4Dプロセスまたは RESTセッション) によって異なります。
			4Dプロセスによるロックの場合:
task_id	number	プロセスID	
user_name	テキスト	マシン上でのセッションユーザー名	
user4d_alias	テキスト	4D ユーザーの名前またはエイリアス	
user4d_id	number	4DデータベースディレクトリでのユーザーID	
host_name	テキスト	マシン名	
task_name	テキスト	プロセス名	
client_version	テキスト	クライアントのバージョン	
			RESTセッションによるロックの場合:
host	テキスト	エンティティをロックした URL (例: " www.myserver.com ")	
IPAddr	テキスト	ロック元の IPアドレス (例: "127.0.0.1")	
userAgent	テキスト	ロック元の userAgent (例: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36")	
			深刻なエラーの場合にのみ利用可能 (深刻なエラーとは、プライマリーキーを重複させようとした、ディスクがいっぱいであった、などです):
errors		Object の Collection	
	message	テキスト	エラーメッセージ
	component signature	テキスト	内部コンポーネント署名 (例 "dmbg" はデータベースコンポーネントを表します)
	errCode	number	エラーコード

(*) エラー時には *Result* オブジェクトの *status* あるいは *statusText* プロパティに以下のいずれかの値が返されます:

定数	値	説明
dk status entity does not exist anymore	5	<p>エンティティはもうデータ内に存在していません。このエラーは以下のような場合に起きます:</p> <ul style="list-style-type: none"> エンティティがドロップされている (スタンプが変更されていて、メモリ空間は解放されている) エンティティがドロップされていて、他のプライマリーキー値を持つエンティティで置き換えられている (スタンプは変更されていて、新しいエンティティがメモリ空間を使用している)。entity.drop() を使用するとき、このエラーは dk force drop if stamp changed オプションを使用した場合に返されることがあります。entity.lock() を使用するとき、このエラーは dk reload drop if stamp changed オプションを使用した場合に返されることがあります。 <p>割り当てられた statusText : "エンティティはもう存在しません"</p>
dk status locked	3	<p>エンティティはペシミスティック・ロックでロックされています。</p> <p>割り当てられた statusText : "既にロックされています"</p>
dk status serious error	4	<p>深刻なエラーとは、低レベルのデータベースエラー (例: 重複キー)、ハードウェアエラーなどです。</p> <p>割り当てられた statusText : "その他のエラー"</p>
dk status stamp has changed	2	<p>エンティティの内部的なスタンプ値がデータ内に保存されているエンティティのものと合致しません (オptyimistic lock)。</p> <ul style="list-style-type: none"> entity.save() の場合: dk auto merge オプションが使用されていない場合に限りエラー entity.drop() の場合: dk force drop if stamp changed オプションが使用されていない場合に限りエラー entity.lock() の場合: dk reload if stamp changed オプションが使用されていない場合に限りエラー <p>割り当てられた statusText : "スタンプが変更されています"</p>

例題 1

エラーのある例題:

```
var $employee : cs.EmployeeEntity
var $status : Object
$employee:=ds.Employee.get(716)
$status:=$employee.lock()
Case of
  :($status.success)
    ALERT($employee.firstName+" "+$employee.lastName+"をロックしました")
  :($status.status=dk status stamp has changed)
    ALERT($status.statusText)
End case
```

例題 2

dk reload if stamp changed オプションを使用する例:

```
var $employee : cs.EmployeeEntity
var $status : Object
$employee:=ds.Employee.get(717)
$status:=$employee.lock(dk reload if stamp changed)
Case of
  :($status.success)
    ALERT($employee.firstName+" "+$employee.lastName+"をロックしました")
  :($status.status=dk status entity does not exist anymore)
    ALERT($status.statusText)
End case
```

.next()

▶ 履歴

.next() : 4D.Entity

引数	タイプ		説明
戻り値	4D.Entity	<-	エンティティセレクション内の次のエンティティへの参照 (見つからなければ null)

説明

.next() 関数は、エンティティが所属するエンティティセレクションの次のエンティティへの参照を返します。

対象エンティティが所属する既存エンティティセレクションが存在しない場合 (つまり `entity.getSelection()` が Null を返す場合)、関数は Null 値を返します。

エンティティセレクション内に有効な次のエンティティが存在しない場合 (セレクションの最終エンティティの場合)、関数は Null を返します。次のエンティティがドロップされていた場合、関数はその次の有効なエンティティを返します (セレクションの最後に辿り着くと Null を返します)。

例題

```
var $employees : cs.EmployeeSelection
var $employee; $nextEmployee : cs.EmployeeEntity
$employees:=ds.Employee.query("lastName = :1";"H@") // このエンティティセレクションは 3件のエンティティを持ちます
$employee:=$employees[0]
$nextEmployee:=$employee.next() // $nextEmployee は、$employees エンティティセレクションの 2番目のエンティティで
```

.previous()

▶ 履歴

.previous() : 4D.Entity

引数	タイプ		説明
戻り値	4D.Entity	<-	エンティティセレクション内の前のエンティティへの参照 (見つからなければ null)

説明

.previous() 関数は、エンティティが所属するエンティティセレクションの前のエンティティへの参照を返します。

対象エンティティが所属する既存エンティティセレクションが存在しない場合 (つまり `entity.getSelection()` が Null を返す場合)、関数は Null 値を返します。

エンティティセレクション内に有効な前のエンティティが存在しない場合 (セレクションの先頭エンティティの場合)、関数は Null を返します。前のエンティティがドロップされていた場合、関数はその前の有効なエンティティを返します (セレクションの先頭に辿り着くと Null を返します)。

例題

```
var $employees : cs.EmployeeSelection
var $employee; $previousEmployee : cs.EmployeeEntity
$employees:=ds.Employee.query("lastName = :1";"H@") // このエンティティセレクションは 3件のエンティティを持ちます
$employee:=$employees[1]
$previousEmployee:=$employee.previous() // $previousEmployee は、$employees エンティティセレクションの先頭エン
```

.reload()

▶ 履歴

.reload() : Object

引数	タイプ		説明
戻り値	オブジェクト	<-	ステータスオブジェクト

説明

.reload() 関数は、データストアのデータクラスに対応するテーブルに保存されている情報に応じて、エンティティの中身をメモリ内にリロードします。エンティティが同じプライマリーキーで存在している場合にのみリロードは実行されます。

戻り値

.reload() によって返されるオブジェクトには以下のプロパティが格納されます:

プロパティ	タイプ	説明
success	boolean	リロードが成功した場合には true、それ以外は false エラーの場合にのみ利用可能:
status(*)	number	エラーコード、以下参照
statusText(*)	テキスト	エラーの詳細、以下参照

(*) エラー時には Result オブジェクトの status あるいは statusText プロパティに以下のいずれかの値が返されます:

定数	値	説明
dk status entity does not exist anymore	5	<p>エンティティはもうデータ内に存在していません。このエラーは以下のような場合に起きます:</p> <ul style="list-style-type: none"> エンティティがドロップされている (スタンプが変更されていて、メモリ空間は解放されている) エンティティがドロップされていて、他のプライマリーキー値を持つエンティティで置き換えられている (スタンプは変更されていて、新しいエンティティがメモリ空間を使用している)。entity.drop() を使用するとき、このエラーは dk force drop if stamp changed オプションを使用した場合に返されることがあります。entity.lock() を使用するとき、このエラーは dk reload drop if stamp changed オプションを使用した場合に返されることがあります。 <p>割り当てられた statusText : "エンティティはもう存在しません"</p>
dk status serious error	4	<p>深刻なエラーとは、低レベルのデータベースエラー (例: 重複キー)、ハードウェアエラーなどです。</p> <p>割り当てられた statusText : "その他のエラー"</p>

例題

```

var $employee : cs.EmployeeEntity
var $employees : cs.EmployeeSelection
var $result : Object

$employees:=ds.Employee.query("lastName=:1";"Hollis")
$employee:=$employees[0]
$employee.firstName:="Mary"
$result:=$employee.reload()

Case of
    :($result.success)
        ALERT("エンティティはリロードされました")
    :($result.status=dk status entity does not exist anymore)
        ALERT("エンティティはドロップされています")
End case

```

.save()

▶履歴

.save({ mode : Integer }) : Object

引数	タイプ		説明
mode	整数	->	dk auto merge : 自動マージモードを有効化します
戻り値	オブジェクト	<-	保存の結果

説明

.save() 関数は、データクラスに対応するテーブル内に、エンティティの変更内容を保存します。エンティティを作成したあと、あるいはエンティティに対して保存したい変更をおこなったあとにはこの関数を呼び出す必要があります。

保存処理は、少なくとも一つのエンティティ属性が "touched" である（更新されている）場合にのみ実行されます（.touched() および .touchedAttributes() 関数参照）。そうでない場合、関数は何もしません（トリガーは呼び出されません）。

マルチユーザー、あるいはマルチプロセスアプリケーションにおいて、.save() 関数は "オプティミスティック・ロック" 機構のもとで実行されます。これはコードが保存されるたびに内部的なロックスタンプが自動的に増分していくという機構です。

mode 引数を渡さなかった場合のデフォルトでは、いずれの属性に関わらず同エンティティが他のプロセスまたはユーザーによって変更されていた場合にエラーを返します（以下参照）。

mode に dk auto merge オプションを渡すと自動マージモードが有効化され、別のプロセス/ユーザーが同エンティティに対して同時に変更をおこなっていても、異なる属性に対する変更であればエラーは生成されません。エンティティに保存されるデータは、別々の変更処理の組み合わせ（"マージ（併合）"）になります（同じ属性に対して変更がおこなわれた場合には、自動マージモードであっても保存は失敗し、エラーが返されます）。

ピクチャー・オブジェクト・テキスト型属性で、データを外部保存している場合には、自動マージモードは利用できません。これらの属性に同時の変更があった場合には dk status stamp has changed エラーになります。

戻り値

.save() によって返されるオブジェクトには以下のプロパティが格納されます：

プロパティ		タイプ	説明
success		boolean	保存に成功した場合には true、それ以外は false
			<i>dk auto merge</i> オプションが使用されていた場合にのみ利用可能：
autoMerged		boolean	自動マージが実行された場合には true、それ以外は false
			エラーの場合にのみ利用可能：
status		number	エラーコード、 以下参照
statusText		テキスト	エラーの詳細、 以下参照
			ペシミスティック・ロックエラーの場合にのみ利用可能：
lockKindText		テキスト	"Locked by record"
lockInfo		object	ロック元についての情報
	task_id	number	プロセスID
	user_name	テキスト	マシン上でのセッションユーザー名
	user4d_alias	テキスト	SET USER ALIAS で設定されていればユーザーイリアス。それ以外は 4D ディレクトリのユーザー名
	host_name	テキスト	マシン名
	task_name	テキスト	プロセス名
	client_version	テキスト	
			深刻なエラーの場合にのみ利用可能（深刻なエラーとは、プライマリーキーを重複させようとした、ディスクがいっぱいであった、などです）：
errors		Object の Collection	
	message	テキスト	エラーメッセージ
	componentSignature	テキスト	内部コンポーネント署名（例 "dmbg" はデータベースコンポーネントを表します）
	errCode	number	エラーコード

status と statusText

エラー時には Result オブジェクトの `status` あるいは `statusText` プロパティに以下のいずれかの値が返されます：

定数	値	説明
dk status automerge failed	6	(dk auto merge オプションが使用されたときのみ) エンティティを保存するときに自動マージオプションが失敗しました。 割り当てられた statusText : "自動マージ失敗"
dk status entity does not exist anymore	5	エンティティはもうデータ内に存在していません。このエラーは以下のような場合に起きます: <ul style="list-style-type: none"> エンティティがドロップされている (スタンプが変更されていて、メモリ空間は解放されている) エンティティがドロップされていて、他のプライマリーキー値を持つエンティティで置き換えられている (スタンプは変更されていて、新しいエンティティがメモリ空間を使用している)。 entity.drop() を使用するとき、このエラーは dk force drop if stamp changed オプションを使用した場合に返されることがあります。 entity.lock() を使用するとき、このエラーは dk reload drop if stamp changed オプションを使用した場合に返されることがあります。 割り当てられた statusText : "エンティティはもう存在しません"
dk status locked	3	エンティティはペシミスティック・ロックでロックされています。 割り当てられた statusText : "既にロックされています"
dk status serious error	4	深刻なエラーとは、低レベルのデータベースエラー (例: 重複キー)、ハードウェアエラーなどです。 割り当てられた statusText : "その他のエラー"
dk status stamp has changed	2	エンティティの内部的なスタンプ値がデータ内に保存されているエンティティのものと合致しません (オプティミスティック・ロック)。 <ul style="list-style-type: none"> entity.save() の場合: dk auto merge オプションが使用されていない場合に限りエラー entity.drop() の場合: dk force drop if stamp changed オプションが使用されていない場合に限りエラー entity.lock() の場合: dk reload if stamp changed オプションが使用されていない場合に限りエラー 割り当てられた statusText : "スタンプが変更されています"

例題 1

新しいエンティティを作成します:

```
var $status : Object
var $employee : cs.EmployeeEntity
$employee:=ds.Employee.new()
$employee.firstName:="Mary"
$employee.lastName:="Smith"
$status:=$employee.save()
If($status.success)
    ALERT("Employee が作成されました")
End if
```

例題 2

dk auto merge オプションを使わずにエンティティを更新します:

```

var $status : Object
var $employee : cs.EmployeeEntity
var $employees : cs.EmployeeSelection
$employees:=ds.Employee.query("lastName=:1";"Smith")
$employee:=$employees.first()
$employee.lastName:="Mac Arthur"
$status:=$employee.save()
Case of
    :($status.success)
        ALERT("Employee が更新されました")
    :($status.status=dk status stamp has changed)
        ALERT($status.statusText)
End case

```

例題 3

`dk auto merge` オプションを使ってエンティティを更新します:

```

var $status : Object
var $employee : cs.EmployeeEntity
var $employees : cs.EmployeeSelection

$employees:=ds.Employee.query("lastName=:1";"Smith")
$employee:=$employees.first()
$employee.lastName:="Mac Arthur"
$status:=$employee.save(dk auto merge)
Case of
    :($status.success)
        ALERT("Employee を更新しました")
    :($status.status=dk status automerge failed)
        ALERT($status.statusText)
End case

```

.toObject()

▶ 履歴

`.toObject() : Object`
`.toObject(filterString : Text { ; options : Integer}) : Object`
`.toObject(filterCol : Collection { ; options : Integer }) : Object`

引数	タイプ		説明
filterString	テキスト	->	取得する属性 (カンマ区切り)
filterCol	コレクション	->	取得する属性のコレクション
options	整数	->	<code>dk with primary key : __KEY</code> プロパティを追加; <code>dk with stamp : __STAMP</code> プロパティを追加
戻り値	オブジェクト	<-	エンティティを元にビルトされたオブジェクト

説明

`.toObject()` 関数は、エンティティからビルトされたオブジェクトを返します。オブジェクト内部のプロパティ名はエンティティの属性名と合致します。

`filterString` 引数が空の文字列、あるいは "*" の場合、以下のいずれかが返されます:

- すべてのストレージエンティティ属性
- リレートエンティティ型の属性 (`kind` が `relatedEntity`) : リレートエンティティと同じ名前 (N対1リレーション名) のプロパティ。属性は単純な形式で取得されます。

- リレートエンティティズ型の属性 (`kind` が `relatedEntities`): 属性は返されません。

最初の引数として、取得するエンティティ属性を渡します。以下のものを渡すことができます:

- `filterString`: プロパティパスをカンマで区切った文字列: "propertyPath1, propertyPath2, ... " または
- `filterCol`: 文字列のコレクション: ["propertyPath1", "propertyPath2", ...]

`filter` 引数がリレートエンティティ型の属性を指定する場合:

- `propertyPath = "relatedEntity" -> 單純な形式で取得されます: __KEY プロパティ(プライマリーキー)を持つオブジェクト`
- `propertyPath = "relatedEntity.*" -> 全プロパティが取得されます`
- `propertyPath = "relatedEntity.propertyName1; relatedEntity.propertyName2; ..." -> 指定されたプロパティのみが取得されます`

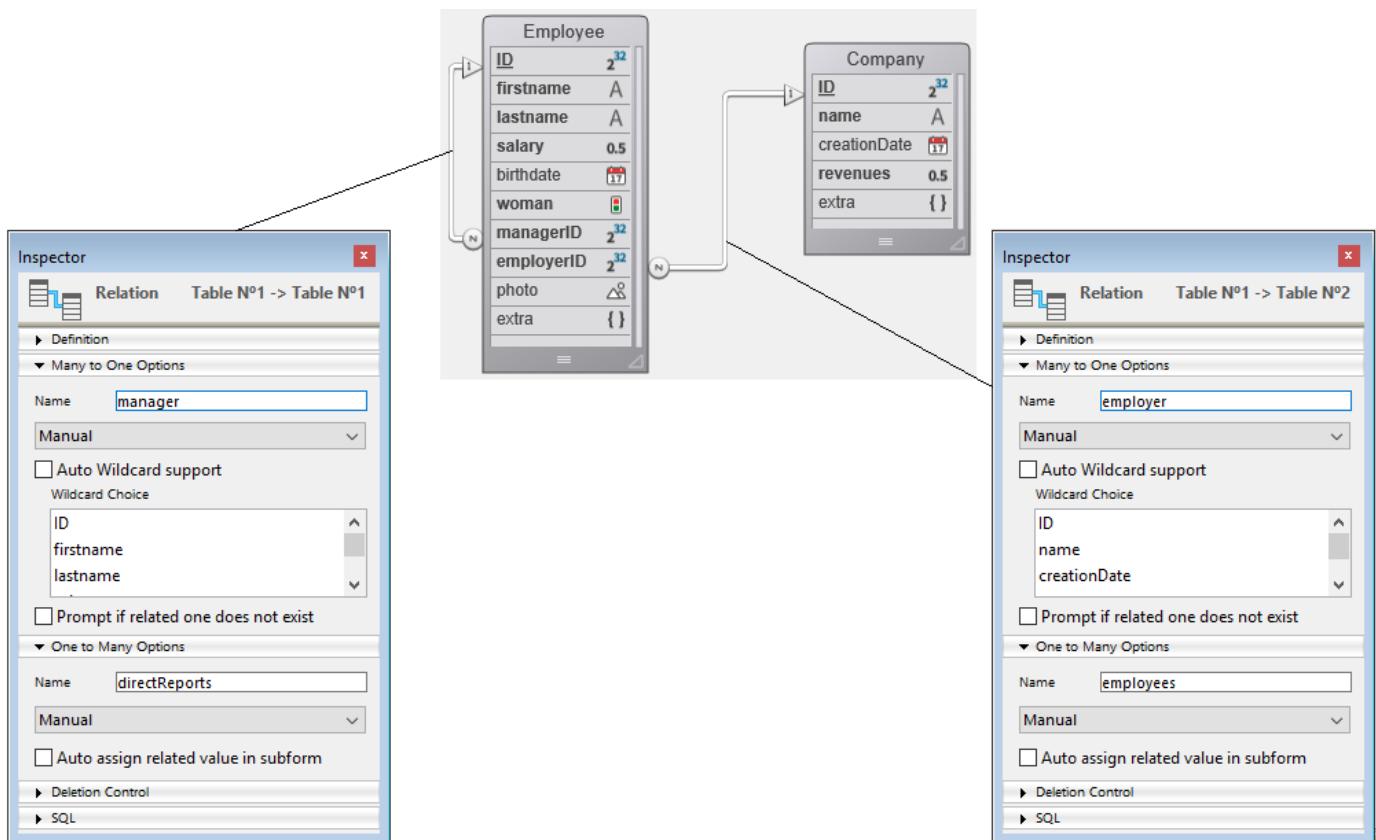
`filter` 引数がリレートエンティティズ型の属性を指定する場合:

- `propertyPath = "relatedEntities.*" -> 全プロパティが取得されます`
- `propertyPath = "relatedEntities.propertyName1; relatedEntities.propertyName2; ..." -> 指定されたプロパティのみが取得されます`

`options` に `dk with primary key` または `dk with stamp` セレクターを渡すことで、エンティティのプライマリーキー/スタンプを、取得するオブジェクトに追加するかどうかを指定できます。

例題 1

このセクションの例題では、以下のストラクチャーを使います:



`filter` 引数を渡さない場合:

```
employeeObject:=employeeSelected.toObject()
```

戻り値:

```
{
    "ID": 413,
    "firstName": "Greg",
    "lastName": "Wahl",
    "salary": 0,
    "birthDate": "1963-02-01T00:00:00.000Z",
    "woman": false,
    "managerID": 412,
    "employerID": 20,
    "photo": "[object Picture]",
    "extra": null,
    "employer": { // 単純な形式で取得されたリレートエンティティ
        "__KEY": 20
    },
    "manager": {
        "__KEY": 412
    }
}
```

例題 2

プライマリーキーとスタンプを取得します:

```
employeeObject:=employeeSelected.toObject("";dk with primary key+dk with stamp)
```

戻り値:

```
{
    "__KEY": 413,
    "__STAMP": 1,
    "ID": 413,
    "firstName": "Greg",
    "lastName": "Wahl",
    "salary": 0,
    "birthDate": "1963-02-01T00:00:00.000Z",
    "woman": false,
    "managerID": 412,
    "employerID": 20,
    "photo": "[object Picture]",
    "extra": null,
    "employer": {
        "__KEY": 20
    },
    "manager": {
        "__KEY": 412
    }
}
```

例題 3

リレートエンティティズのプロパティをすべて展開します:

```
employeeObject:=employeeSelected.toObject("directReports.*")
```

```
{
  "directReports": [
    {
      "ID": 418,
      "firstName": "Lorena",
      "lastName": "Boothe",
      "salary": 44800,
      "birthDate": "1970-10-02T00:00:00.000Z",
      "woman": true,
      "managerID": 413,
      "employerID": 20,
      "photo": "[object Picture]",
      "extra": null,
      "employer": {
        "__KEY": 20
      },
      "manager": {
        "__KEY": 413
      }
    },
    {
      "ID": 419,
      "firstName": "Drew",
      "lastName": "Caudill",
      "salary": 41000,
      "birthDate": "2030-01-12T00:00:00.000Z",
      "woman": false,
      "managerID": 413,
      "employerID": 20,
      "photo": "[object Picture]",
      "extra": null,
      "employer": {
        "__KEY": 20
      },
      "manager": {
        "__KEY": 413
      }
    },
    {
      "ID": 420,
      "firstName": "Nathan",
      "lastName": "Gomes",
      "salary": 46300,
      "birthDate": "2010-05-29T00:00:00.000Z",
      "woman": false,
      "managerID": 413,
      "employerID": 20,
      "photo": "[object Picture]",
      "extra": null,
      "employer": {
        "__KEY": 20
      },
      "manager": {
        "__KEY": 413
      }
    }
  ]
}
}
```

例題 4

リレートエンティティズの一部のプロパティを取得します:

```
employeeObject:=employeeSelected.toObject("firstName, directReports.lastName")
```

戻り値:

```
{  
    "firstName": "Greg",  
    "directReports": [  
        {  
            "lastName": "Boothe"  
        },  
        {  
            "lastName": "Caudill"  
        },  
        {  
            "lastName": "Gomes"  
        }  
    ]  
}
```

例題 5

リレートエンティティを単純な形式で取得します:

```
$coll:=New collection("firstName";"employer")  
employeeObject:=employeeSelected.toObject($coll)
```

戻り値:

```
{  
    "firstName": "Greg",  
    "employer": {  
        "__KEY": 20  
    }  
}
```

例題 6

リレートエンティティの全プロパティを取得します:

```
employeeObject:=employeeSelected.toObject("employer.*")
```

戻り値:

```
{  
    "employer": {  
        "ID": 20,  
        "name": "India Astral Secretary",  
        "creationDate": "1984-08-25T00:00:00.000Z",  
        "revenues": 12000000,  
        "extra": null  
    }  
}
```

例題 7

リレートエンティティの一部のプロパティを取得します:

```
$col:=New collection  
$col.push("employer.name")  
$col.push("employer.revenues")  
employeeObject:=employeeSelected.toObject($col)
```

戻り値:

```
{  
    "employer": {  
        "name": "India Astral Secretary",  
        "revenues": 12000000  
    }  
}
```

.touched()

▶ 覆歴

.touched() : Boolean

引数	タイプ		説明
戻り値	ブール	<-	少なくとも一つのエンティティ属性が編集されていて未保存の場合に true、それ以外の場合には false

説明

.touched() 関数は、エンティティがメモリに読み込まれてから、あるいは保存されてから、エンティティ属性が変更されたかどうかをテストします。

属性が更新あるいは計算されていた場合、関数は true を返し、それ以外は false を返します。この関数を使用することで、エンティティを保存する必要があるかどうかを確認することができます。

この関数は、(.new()) で作成された) 新規エンティティに対しては常に false を返します。ただし、エンティティの属性を計算する関数を使用した場合には、.touched() 関数は true を返します。たとえば、プライマリーキーを計算するために .getKey() を呼び出した場合、.touched() メソッドは true を返します。

例題

エンティティを保存する必要があるかどうかをチェックします:

```
var $emp : cs.EmployeeEntity  
$emp:=ds.Employee.get(672)  
$emp.firstName:=$emp.firstName // 同じ値で更新されても、属性は変更されたと判定されます  
  
If($emp.touched()) // 一つ以上の属性が変更されていた場合  
    $emp.save()  
End if // そうでない場合はエンティティを保存する必要はありません
```

.touchedAttributes()

▶ 覆歴

.touchedAttributes() : Collection

引数	タイプ		説明
戻り値	コレクション	<-	変更された属性の名前、あるいは空のコレクション

説明

`.touchedAttributes()` 関数は、メモリに読み込み後に変更されたエンティティの属性名を返します。

この関数は、種類 (`kind`) が `storage` あるいは `relatedEntity` である属性に適用されます。

リレート先のエンティティそのものが更新されていた場合 (外部キーの変更)、リレートエンティティの名称とそのプライマリーキー名が返されます。

エンティティ属性がいずれも更新されていなかった場合、関数は空のコレクションを返します。

例題 1

```
var $touchedAttributes : Collection
var $emp : cs.EmployeeEntity

$touchedAttributes:=New collection
$emp:=ds.Employee.get(725)
$emp.firstName:=$emp.firstName // 同じで値で更新されていても、属性は変更されたと判定されます
$emp.lastName:="Martin"
$touchedAttributes:=$emp.touchedAttributes()
// $touchedAttributes: ["firstName", "lastName"]
```

例題 2

```
var $touchedAttributes : Collection
var $emp : cs.EmployeeEntity
var $company : cs.CompanyEntity

$touchedAttributes:=New collection

$emp:=ds.Employee.get(672)
$emp.firstName:=$emp.firstName
$emp.lastName:="Martin"

$company:=ds.Company.get(121)
$emp.employer:=$company

$touchedAttributes:=$emp.touchedAttributes()

// $touchedAttributes コレクション: ["firstName", "lastName", "employer", "employerID"]
```

この場合において:

- `firstName` および `lastName` はストレージ (`storage`) 型です
- `employer` はリレートエンティティ (`relatedEntity`) 型です
- `employerID` は、`employer` リレートエンティティの外部キーです

.unlock()

▶ 履歴

`.unlock()` : Object

引数	タイプ		説明
戻り値	オブジェクト	<-	ステータスオブジェクト

説明

`.unlock()` 関数は、データストアおよび、データクラスに対応するテーブル内の、対象エンティティが参照するレコードのペシミスティック・ロックを解除します。

詳細については [エンティティロック](#) を参照ください。

ロックしているプロセス内のどのエンティティからもレコードが参照されなくなった場合、自動的にレコードロックが解除されます（たとえば、エンティティのローカル参照に対してのみロックがかかっていた場合、プロセスが終了すればエンティティおよびレコードのロックは解除されます）。

レコードがロックされている場合、ロックしているプロセスから、ロックされたエンティティ参照に対してロックを解除する必要があります：たとえば：

```
$e1:=ds.Emp.all()[0]
$e2:=ds.Emp.all()[0]
$res:=$e1.lock() // $res.success=true
$res:=$e2.unlock() // $res.success=false
$res:=$e1.unlock() // $res.success=true
```

戻り値

`.unlock()` によって返されるオブジェクトには以下のプロパティが格納されます：

プロパティ	タイプ	説明
success	Boolean	ロック解除が成功した場合には true、それ以外は false ドロップされたエンティティや、ロックされてないレコード、あるいは他のプロセスや他のエンティティによってロックされたレコードに対してロック解除を実行した場合、success には false が返されます。

例題

```
var $employee : cs.EmployeeEntity
var $status : Object

$employee:=ds.Employee.get(725)
$status:=$employee.lock()
... // 処理
$status:=$employee.unlock()
If($status.success)
    ALERT("エンティティのロックは解除されました。")
End if
```

EntitySelection

エンティティセレクションとは、同じ [データクラス](#) に所属する一つ以上の [エンティティ](#) への参照を格納しているオブジェクトのことです。エンティティセレクションは、データクラスから 0個、1個、あるいは X個のエンティティを格納することができます (X はデータクラスに格納されているエンティティの総数です)。

`.all()`、`.query()` などの [DataClass クラス](#) の関数や、`.and()`、`orderBy()` など [EntitySelectionClass](#) クラス自身の関数を用いて、既存のセレクションからエンティティセレクションを作成することができます。また、`dataClass.newSelection()` 関数または `Create entity selection` コマンドを使用して、空のエンティティセレクションを作成することもできます。

概要

[index] : 4D.Entity

標準のコレクションシンタックスを使用してエンティティセレクション内のエンティティにアクセスすることができます

.attributeName : Collection

.attributeName : 4D.EntitySelection

エンティティセレクション内の属性値の "投影" を返します

.add(entity : 4D.Entity) : 4D.EntitySelection

`entity` に渡したエンティティをエンティティセレクションに追加し、編集されたエンティティセレクションを返します

.and(entity : 4D.Entity) : 4D.EntitySelection

.and(entitySelection : 4D.EntitySelection) : 4D.EntitySelection

エンティティセレクションと `entity` あるいは `entitySelection` 引数をAND論理演算子を使用して結合します

.average(attributePath : Text) : Real

`attributePath` に指定した、エンティティセレクション内の null でない値の算術平均 (相加平均) を返します

.contains(entity : 4D.Entity) : Boolean

エンティティ参照がエンティティセレクションに属している場合には `true` を返します

.count(attributePath : Text) : Real

エンティティセレクション内で `attributePath` に指定したパスの値が null でないエンティティの数を返します

.distinct(attributePath : Text { ; option : Integer }) : Collection

`attributePath` に指定した、エンティティセレクション内の重複しない (異なる) 値のみを格納したコレクションを返します

.drop({ mode : Integer }) : 4D.EntitySelection

データストアのデータクラスに対応するテーブルから、エンティティセレクションに所属しているエンティティを削除します

.extract(attributePath : Text { ; option : Integer }) : Collection

.extract(attributePath { ; targetPath } { ; ...attributePathN : Text ; targetPathN : Text }) : Collection

`attributePath` で指定した値をエンティティセレクションから抽出し、コレクションに格納して返します

.first() : 4D.Entity

エンティティセレクションの先頭エンティティへの参照を返します

.getDataClass() : 4D.DataClass

エンティティセレクションのデータクラスを返します

.getRemoteContextAttributes() : Text

エンティティによって使われている最適化コンテキストの情報を返します

.isAlterable() : Boolean

エンティティが変更可能な場合に `true` を返す。逆

エンティティセレクションの属性値を返します	
.isOrdered() : Boolean	エンティティセレクションが順列ありであれば true を返します
.last() : 4D.Entity	エンティティセレクションの最終エンティティへの参照を返します
.length : Integer	エンティティセレクション内のエンティティの数を返します
.max(attributePath : Text) : any	attributePath に指定したエンティティセレクションの属性値のうち最高の (あるいは最大の) 値を返します
.min(attributePath : Text) : any	attributePath に指定したエンティティセレクションの属性値のうち最低の (あるいは最小の) 値を返します
.minus(entity : 4D.Entity) : 4D.EntitySelection	
.minus(entitySelection : 4D.EntitySelection) : 4D.EntitySelection	元のエンティティセレクションから、entity 引数のエンティティ、あるいはentitySelection 引数のエンティティセレクションに含まれるエンティティを除外し、結果のエンティティセレクションを返します
.or(entity : 4D.Entity) : 4D.EntitySelection	
.or(entitySelection : 4D.EntitySelection) : 4D.EntitySelection	OR論理演算子を使用して、entity または entitySelection のエンティティと対象エンティティセレクションを組み合わせます
.orderBy(pathString : Text) : 4D.EntitySelection	
.orderBy(pathObjects : Collection) : 4D.EntitySelection	エンティティセレクションの全エンティティが pathString または pathObjects が指定する順番に並べ替えられた、新規の順列ありのエンティティセレクションを返します
.orderByFormula(formulaString : Text { ; sortOrder : Integer } { ; settings : Object }) : 4D.EntitySelection	
.orderByFormula(formulaObj : Object { ; sortOrder : Integer } { ; settings : Object }) : 4D.EntitySelection	順列ありの新規エンティティセレクションを返します
.query(queryString : Text { ; ...value : any } { ; querySettings : Object }) : 4D.EntitySelection	
.query(formula : Object { ; querySettings : Object }) : 4D.EntitySelection	エンティティセレクションの全エンティティから、queryString または formula と任意の value 引数で指定した検索条件に合致するエンティティを検索します
.queryPath : Text	実際に 4Dで実行されたクエリの詳細な情報
.queryPlan : Text	実行前のクエリの詳細な情報 (クエリプラン) を格納します
.refresh()	ローカルの ORDAキャッシュにあるエンティティセレクションデータを即座に "無効化" します
.selected(selectedEntities : 4D.EntitySelection) : Object	
.slice(startFrom : Integer { ; end : Integer }) : 4D.EntitySelection	エンティティセレクションの一部を、新規エンティティセレクションとして返します
.sum(attributePath : Text) : Real	attributePath に指定したエンティティセレクションの属性値の総和を返します
.toCollection({ options : Inteaer { : beain : Inteaer { : howManv : Inteaer } } }) : Collection	

```
.toCollection( filterString : Text {; options : Integer {; begin : Integer {; howMany : Integer }}} ) : Collection
.toCollection( filterCol : Collection {; options : Integer {; begin : Integer {; howMany : Integer }}} ) : Collection
プロパティと値のセットを持つオブジェクト要素を格納するコレクションを作成し、返します
```

Create entity selection

Create entity selection (*dsTable* : Table { ; *settings* : Object }) : 4D.EntitySelection

引数	タイプ		説明
<i>dsTable</i>	テーブル	->	エンティティセレクションの元となるカレントセレクションが属する 4Dデータベースのテーブル
<i>settings</i>	Object	->	ビルドオプション: context
戻り値	4D.EntitySelection	<-	指定したテーブルに対応するデータクラスのエンティティセレクション

説明

`Create entity selection` コマンドは、*dsTable* で指定したテーブルに対応するデータクラスの [追加可能な](#) 新規エンティティセレクションを、同テーブルのカレントセレクションに基づいてビルトして返します。

ソートされたカレントセレクションの場合、[順列のある](#) エンティティセレクションが作成されます（カレントセレクションの並び順が受け継がれます）。カレントセレクションがソートされていない場合、順列のないエンティティセレクションが作成されます。

ds において *dsTable* が公開されていない場合には、エラーが返されます。リモートデータストアの場合は、このコマンドは使用できません。

任意の *settings* には、以下のプロパティを持つオブジェクトを渡せます：

プロパティ	タイプ	説明
<i>context</i>	Text	エンティティセレクションに適用されている 最適化コンテキスト のラベル。

例題

```
var $employees : cs.EmployeeSelection
ALL RECORDS([Employee])
$employees:=Create entity selection([Employee])
// $employees エンティティセレクションには、
// Employee データクラスの全エンティティへの参照が格納されています
```

参照

`dataClass.newSelection()`

USE ENTITY SELECTION

USE ENTITY SELECTION (*entitySelection*)

引数	タイプ		説明
<i>entitySelection</i>	EntitySelection	->	エンティティセレクション

説明

`USE ENTITY SELECTION` コマンドは、*entitySelection* 引数で指定したデータクラスに合致するテーブルのカレントセレクションを、エンティティセレクションの中身に応じて更新します。

[リモートデータストア](#) の場合は、このコマンドは使用できません。

`USE ENTITY SELECTION` の呼び出し後、更新された（空でない）カレントセレクションの最初のレコードがカレントレコードとなります。それにはメモリ内にはロードされません。カレントレコードのフィールド値を使用するには、`USE ENTITY SELECTION` コマンドの後に `LOAD RECORD` コマンドを使用します。

例題

```
var $entitySel : Object  
  
$entitySel:=ds.Employee.query("lastName = :1";"M@") // $entitySel は Employee データクラスにリレートされています。  
REDUCE SELECTION([Employee];0)  
USE ENTITY SELECTION($entitySel) // Employee テーブルのカレントセレクションが更新されました
```

[index]

▶ 覆歴

[index] : 4D.Entity

説明

`EntitySelection[index]` 記法を使用すると、標準のコレクションシンタックスを使用してエンティティセレクション内のエンティティにアクセスすることができます。取得したいエンティティの位置を `index` に渡します。

対応するエンティティはデータストアから再読み込みされる点に注意してください。

`index` には、0 と `.length -1` の範囲内で数値を指定することができます。

- `index` が範囲外だった場合、エラーが返されます。
- `index` がドロップされたエンティティに対応していた場合、Null値が返されます。

警告: `EntitySelection[index]` は代入不可の式です。これは、`.lock()` や `.save()` などの関数において、編集可能なエンティティ参照として使用することはできない、ということを意味します。エンティティを操作するには、戻り値を変数などの代入可能な式に割り当てる必要があります。例:

```
$sel:=ds.Employee.all() // エンティティセレクションを作成  
// 無効なコード:  
$result:=$sel[0].lock() //動作しません  
$sel[0].lastName:="Smith" //動作しません  
$result:=$sel[0].save() //動作しません  
// 有効なコード:  
$entity:=$sel[0] //OK  
$entity.lastName:="Smith" //OK  
$entity.save() //OK
```

例題

```
var $employees : cs.EmployeeSelection  
var $employee : cs.EmployeeEntity  
$employees:=ds.Employee.query("lastName = :1";"H@")  
$employee:=$employees[2] // $employees エンティティセレクションの3番目のエンティティがデータベースからリロードされま
```

`.attributeName`

▶ 履歴

.attributeName : Collection
.attributeName : 4D.EntitySelection

説明

データクラス属性はすべてエンティティセレクションのプロパティとして利用可能で、エンティティセレクション内の属性値の "投影" を返します。戻り値は、属性の種類 (`kind` が `storage` あるいは `relation`) によって、コレクションあるいは新しいエンティティセレクションのどちらかになります。

- `attributeName` で指定した属性がストレージ型の場合: `.attributeName` は `attributeName` と同じ型の値のコレクションを返します。
- `attributeName` で指定した属性がリレートエンティティ型の場合: `.attributeName` は `attributeName` と同じ型のリレート値の新規エンティティセレクションを返します。重複しているエンティティは取り除かれます (返されるのは順列なしのエンティティセレクションです)。
- `attributeName` で指定した属性がリレートエンティティズ型の場合: `.attributeName` は `attributeName` と同じ型のリレート値の新規エンティティセレクションを返します。重複しているエンティティは取り除かれます (返されるのは順列なしのエンティティセレクションです)。

エンティティセレクションのプロパティとしてリレーション属性が使用されると、返される結果は、たとえ返されるエンティティが一つだけだとしても、常に新しいエンティティセレクションとなります。エンティティが何も返ってこない場合には、返されるのは空のエンティティセレクションです。

属性がエンティティセレクション内に存在しない場合、エラーが返されます。

例題 1

ストレージ値の投影:

```
var $firstNames : Collection
$entitySelection:=ds.Employee.all()
$firstNames:=$entitySelection.firstName // firstName は文字列型です
```

返されるのは文字列のコレクションとなります。例:

```
[  
  "Joanna",  
  "Alexandra",  
  "Rick"  
]
```

例題 2

リレートエンティティの投影:

```
var $es; $entitySelection : cs.EmployeeSelection
$entitySelection:=ds.Employee.all()
$es:=$entitySelection.employer // employer は Company データクラスにリレートされています
```

返されるオブジェクトは、重複してあるもの (あれば) を取り除いた、Company のエンティティセレクションです。

例題 3

リレートエンティティズの投影:

```
var $es : cs.EmployeeSelection
$es:=ds.Employee.all().directReports // directReports は Employee データクラスにリレートされています
```

返されるオブジェクトは、重複してあるもの (あれば) を取り除いた、Employee のエンティティセレクションです。

`.add()`

▶ 履歴

.add(entity : 4D.Entity) : 4D.EntitySelection

引数	タイプ		説明
entity	4D.Entity	->	エンティティセレクションに追加するエンティティ
戻り値	4D.EntitySelection	->	追加エンティティを含むエンティティセレクション

説明

.add() 関数は、entity に渡したエンティティをエンティティセレクションに追加し、編集されたエンティティセレクションを返します。

このコマンドは、元のエンティティセレクションを変更します。

警告: エンティティセレクションは 追加可能 のものでなければなりません。つまり .newSelection() あるいは Create entity selection などで作成されたものでなければならぬということです。そうでない場合、.add() はエラーを返します。共有可能なエンティティセレクションはエンティティの追加を受け付けないからです。詳細については [共有可能なエンティティセレクション](#) を参照ください。

- エンティティセレクションが順列ありの場合、entity 引数のエンティティはセレクションの最後に追加されます。同じエンティティへの参照がそのエンティティセレクションにすでに所属していた場合、エンティティは重複することになり、同エンティティの新しい参照が追加されます。
- エンティティセレクションが順列なしの場合、entity 引数のエンティティはセレクションの不特定の場所へ追加され、順番付けはされません。

詳細については、[エンティティセレクションの順列あり/順列なし](#) を参照ください。

編集されたエンティティセレクションが関数から返されるため、関数の呼び出しをつなげることができます。

entity 引数のエンティティとエンティティセレクションが同じデータクラスに属していない場合、エラーが発生します。追加するエンティティが Null であった場合には、エラーは発生しません。

例題 1

```
var $employees : cs.EmployeeSelection
var $employee : cs.EmployeeEntity
$employees:=ds.Employee.query("lastName = :1";"S@") // 共有可能なエンティティセレクションです
$employee:=ds.Employee.new()
$employee.lastName:="Smith"
$employee.save()
$employees:=$employees.copy() // 追加可能なエンティティセレクションを取得します
$employees.add($employee) // $employee エンティティが $employees エンティティセレクションへと追加されます
```

例題 2

関数の呼び出しをつないでいくことができます:

```
var $sel : cs.ProductSelection
var $p1;$p2;$p3 : cs.ProductEntity

$p1:=ds.Product.get(10)
$p2:=ds.Product.get(11)
$p3:=ds.Product.get(12)
$sel:=ds.Product.query("ID > 50")
$sel:=$sel.copy()
$sel:=$sel.add($p1).add($p2).add($p3)
```

.and()

▶ 履歴

```
.and( entity : 4D.Entity ) : 4D.EntitySelection  
.and( entitySelection : 4D.EntitySelection ) : 4D.EntitySelection
```

引数	タイプ		説明
entity	4D.Entity	->	交差するエンティティ
entitySelection	4D.EntitySelection	->	交差するエンティティセレクション
戻り値	4D.EntitySelection	<-	AND論理演算子による共通部分の結果を格納する新しいエンティティセレクション

説明

`.and()` 関数は、エンティティセレクションと `entity` あるいは `entitySelection` 引数をAND論理演算子を使用して結合します。戻り値は、エンティティセレクションと引数の両方から参照されているエンティティのみを格納した、順列なしの新規エンティティセレクションです。

- `entity` 引数を渡した場合、引数のエンティティをエンティティセレクションと結合させることになります。エンティティがエンティティセレクションに属している場合、そのエンティティのみを格納する新しいエンティティセレクションが返されます。そうでない場合、空のエンティティセレクションが返されます。
- `entitySelection` 引数を渡した場合、二つのエンティティセレクションを結合させることになります。両方のセレクションから参照されているエンティティのみを格納する新しいエンティティセレクションが返されます。重複するエンティティがなかった場合、空のエンティティセレクションが返されます。

順列ありと順列なしのエンティティセレクション を比較することができます。返されるセレクションは常に順列なしのものになります。

元のエンティティセレクションあるいは `entitySelection` 引数が空であった場合、あるいは `entity` 引数が Null であった場合、空のエンティティセレクションが返されます。

元のエンティティセレクションおよび引数が同じデータクラスのものでない場合、エラーが返されます。

例題 1

```
var $employees; $result : cs.EmployeeSelection  
var $employee : cs.EmployeeEntity  
$employees:=ds.Employee.query("lastName = :1";"H@")  
// $employees エンティティセレクションには、主キー710のエンティティと  
// その他のエンティティが含まれます  
// 例: "Colin Hetrick" / "Grady Harness" / "Sherlock Holmes" (主キー710)  
$employee:=ds.Employee.get(710) // "Sherlock Holmes" を返します  
  
$result:=$employees.and($employee) // $result は主キー710 ("Sherlock Holmes") の  
// エンティティのみを格納するエンティティセレクション。
```

例題 2

"Jones" という名前で、New York に住んでいる従業員のセレクションを取得します:

```
var $sel1; $sel2; $sel3 : cs.EmployeeSelection  
$sel1:=ds.Employee.query("name =:1";"Jones")  
$sel2:=ds.Employee.query("city=:1";"New York")  
$sel3:=$sel1.and($sel2)
```

.average()

▶ 履歴

```
.average( attributePath : Text ) : Real
```

引数	タイプ		説明
attributePath	テキスト	->	計算に使用する属性パス
戻り値	実数	<-	エンティティの属性値の算術平均 (相加平均) (エンティティセレクションからの場合には undefined を返します)

説明

.average() 関数は、attributePath に指定した、エンティティセレクション内の null でない値の算術平均 (相加平均) を返します。

attributePath 引数として、評価する属性パスを渡します。

計算の対象となるのは数値のみです。ただし、エンティティセレクションの attributePath 引数で指定したパスに異なる型の値が混在している場合、.average() はすべてのスカラー要素を対象として平均値を算出します。

日付値は数値 (秒数) に変換され、平均を計算するのに使用されます。

エンティティセレクションが空の場合、または attributePath 引数に数値型の値が含まれていない場合には、.average() は undefined を返します。

以下の場合には、エラーが返されます:

- attributePath はリレート属性である
- attributePath がエンティティセレクションデータクラス内に存在しない属性を指定している場合。

例題

給与が平均より高い従業員の一覧を取得します:

```
var $averageSalary : Real
var $moreThanAv : cs.EmployeeSelection
$averageSalary:=ds.Employee.all().average("salary")
$moreThanAv:=ds.Employee.query("salary > :1";$averageSalary)
```

.contains()

▶履歴

.contains(entity : 4D.Entity) : Boolean

引数	タイプ		説明
entity	4D.Entity	->	評価するエンティティ
戻り値	ブール	<-	エンティティがエンティティセレクションに属している場合には true、そうでない場合は false

説明

.contains() 関数は、エンティティ参照がエンティティセレクションに属している場合には true を返します。そうでない場合には false を返します。

entity 引数として、エンティティセレクション内で検索するエンティティを渡します。エンティティが Null の場合、関数は false を返します。

entity 引数とエンティティセレクションが同じデータクラスのものでない場合、エラーが生成されます。

例題

```

var $employees : cs.EmployeeSelection
var $employee : cs.EmployeeEntity

$employees:=ds.Employee.query("lastName=:1";"H@")
$employee:=ds.Employee.get(610)

If($employees.contains($employee))
    ALERT("主キー610のエンティティのラストネームは H で始まります。")
Else
    ALERT("主キー610のエンティティのラストネームは H で始まりません。")
End if

```

.count()

▶ 覆歴

.count(*attributePath* : Text) : Real

引数	タイプ		説明
<i>attributePath</i>	テキスト	->	計算に使用する属性パス
戻り値	実数	<-	エンティティセレクション内の <i>attributePath</i> が null でない値の個数

説明

.count() 関数は、エンティティセレクション内で *attributePath* に指定したパスの値が null でないエンティティの数を返します。

対象となるのはスカラー値のみです。オブジェクトあるいはコレクション型の値は Null 値とみなされます。

以下の場合には、エラーが返されます:

- *attributePath* はリレート属性である
- *attributePath* がエンティティセレクションデータクラス内に存在しない場合。

例題

ある会社の全従業員のうち、役職のない者を除いた人数を確認します:

```

var $sel : cs.EmployeeSelection
var $count : Real

$sel:=ds.Employee.query("employer = :1";"Acme, Inc")
$count:=$sel.count("jobtitle")

```

.copy()

▶ 覆歴

.copy({ *option* : Integer }) : 4D.EntitySelection

引数	タイプ		説明
<i>option</i>	整数	->	ck shared : 共有可能なエンティティセレクションを返します
戻り値	4D.EntitySelection	<-	エンティティセレクションのコピー

説明

.copy() 関数は、元のエンティティセレクションのコピーを返します。

この関数は、元のエンティティセレクションを変更しません。

`option` パラメーターが省略された場合はデフォルトで、たとえコピー元が共有可能なエンティティセレクションであったとしても、関数はデフォルトで追加可能な（共有不可の）新規エンティティセレクションを返します。共有可能なエンティティセレクションを取得するには、`option` に `ck_shared` 定数を渡します。

詳細については [共有可能なエンティティセレクション](#) を参照ください。

例題

フォームロード時に、商品データを格納するための新規の空エンティティセレクションを作成します：

```
Case of
  :(Form event code=On Load)
    Form.products:=ds.Products.newSelection()
End case
```

このエンティティセレクションに商品を登録したのちに、複数のプロセスでこの商品データを共有するには、`Form.products` を共有可能なエンティティセレクションとしてコピーします：

```
...
// Form.products エンティティセレクションに商品データを登録します
Form.products.add(Form.selectedProduct)

Use(Storage)
  If(Storage.products=null)
    Storage.products:=New shared object()
  End if

  Use(Storage.products)
    Storage.products:=Form.products.copy(ck_shared)
  End use
End use
```

.distinct()

▶ 補足

`.distinct(attributePath : Text { ; option : Integer }) : Collection`

引数	タイプ	説明
attributePath	テキスト	-> 重複しない値を取得する属性のパス
option	整数	-> <code>dk diacritical</code> : アクセント等の発音区別符号を無視しない評価 (たとえば "A" # "a")
戻り値	コレクション	<- 重複しない値のみを格納したコレクション

説明

`.distinct()` 関数は、`attributePath` に指定した、エンティティセレクション内の重複しない（異なる）値のみを格納したコレクションを返します。

返されたコレクションは自動的に並べ替えられています。 Null 値は返されません。

`attributePath` 引数として、固有の値を取得したいエンティティ属性を渡します。スカラー値（テキスト、数値、ブール、あるいは日付）のみが可能です。`attributePath` のパスが異なる型の値を格納しているオブジェクトプロパティであった場合、まず最初に型ごとにグループ分けされ、その後で並べ替えられます。型は以下の順番で返されます：

1. ブール

2. 文字列
3. 数値
4. 日付

`attributePath` がオブジェクト内のパスの場合、`[]` を使ってコレクションを指定できます (例題参照)。

デフォルトでは、アクセント等の発音区別符号を無視した評価が実行されます。評価の際に文字の大小を区別したり、アクセント記号を区別したい場合には、`option` に `dk diacritical` 定数を渡します。

以下の場合には、エラーが返されます:

- `attributePath` はリレート属性である
- `attributePath` がエンティティセレクションデータクラス内に存在しない場合。

例題

国名ごとに重複しない要素を格納するコレクションを取得します:

```
var $countries : Collection  
$countries:=ds.Employee.all().distinct("address.country")
```

`extra` がオブジェクト属性で、`nicknames` がコレクションの場合:

```
$values:=ds.Employee.all().distinct("extra.nicknames[].first")
```

.drop()

▶ 履歴

`.drop({ mode : Integer })` : 4D.EntitySelection

引数	タイプ		説明
mode	整数	->	<code>dk stop dropping on first error</code> : 最初のドロップ不可エンティティで実行を止めます
戻り値	4D.EntitySelection	<-	成功した場合には空のエンティティセレクション、そうでない場合にはドロップ不可エンティティを格納したエンティティセレクション

説明

`.drop()` 関数は、データストアのデータクラスに対応するテーブルから、エンティティセレクションに所属しているエンティティを削除します。エンティティセレクションはメモリ内に残ります。

エンティティの削除は恒久的なものであり、取り消しはできません。ロールバックで戻すことができるよう、この関数はトランザクション内で呼び出すことが推奨されています。

`.drop()` の実行中にロックされたエンティティに遭遇した場合、そのエンティティは削除されません。デフォルトでは、メソッドはエンティティセレクション内のすべてのエンティティを処理し、ドロップ不可なエンティティはエンティティセレクション内に返します。最初のドロップ不可なエンティティに遭遇した時点でメソッドの実行を止める場合は、`mode` パラメーターに `dk stop dropping on first error` 定数を渡します。

例題

`dk stop dropping on first error` オプションを使用しない例:

```

var $employees; $notDropped : cs.EmployeeSelection
$employees:=ds.Employee.query("firstName=:1";"S@")
$notDropped:=$employees.drop() // $notDropped は削除されなかったエンティティをすべて格納したエンティティセレクション
If($notDropped.length=0) // 削除アクションが成功し、すべてのエンティティが削除された場合
    ALERT(String($employees.length)+" 件の社員データを削除しました。") // 削除されたエンティティはメモリの中には残っ
Else
    ALERT("削除中に問題が発生しました。時間をおいて再度お試しください。")
End if

```

`dk stop dropping on first error` オプションを使用する例:

```

var $employees; $notDropped : cs.EmployeeSelection
$employees:=ds.Employee.query("firstName=:1";"S@")
$notDropped:=$employees.drop(dk stop dropping on first error) // $notDropped は削除できなかった最初のエンティ
If($notDropped.length=0) // 削除アクションが成功し、すべてのエンティティが削除された場合
    ALERT(String($employees.length)+" 件の社員データを削除しました。") // 削除されたエンティティはメモリの中には残っ
Else
    ALERT("削除中に問題が発生しました。時間をおいて再度お試しください。")
End if

```

.extract()

▶履歴

`.extract(attributePath : Text { ; option : Integer }) : Collection`
`.extract(attributePath { ; targetPath } { ; ...attributePathN : Text ; targetPathN : Text }) : Collection`

引数	タイプ	説明
attributePath	テキスト	-> 新しいコレクションに抽出する値の属性パス
targetPath	テキスト	-> 抽出先の属性パスあるいは属性名
option	整数	-> <code>ck keep null</code> : 返されるコレクションに <code>null</code> 属性を含めます (デフォルトでは無視されます)。
戻り値	コレクション	<- 抽出した値を格納したコレクション

説明

`.extract()` 関数は、`attributePath` で指定した値をエンティティセレクションから抽出し、コレクションに格納して返します。

`attributePath` には、以下のものを指定することができます:

- スカラーデータクラス属性
- リレートエンティティ (単数)
- リレートエンティティズ (複数)

`attributePath` 引数が無効な場合、空のコレクションが返されます。

このメソッドは 2種類のシンタクスを受け入れます。

`.extract(attributePath : Text { ; option : Integer }) : Collection`

このシンタクスを使用すると、`.extract()` はエンティティセレクションの中の、`attributePath` 引数で指定された値のコレクションを作成して返します。

デフォルトで、`attributePath` で指定された値が `null` または未定義のエンティティは、返されるコレクション内では無視されます。`option` パラメータに `ck keep null` 定数を渡すと、これらの値は返されるコレクションに `null` 要素として格納されます。

- `.kind = "relatedEntity"` であるデータクラス属性は、エンティティのコレクションとして取得されます (重複したものも保持されます)。
- `.kind = "relatedEntities"` であるデータクラス属性は、エンティティセレクションのコレクションとして取得されます。

```
.extract ( attributePath ; targetPath { ; ...attributePathN ; ... targetPathN} ) : Collection
```

このシンタックスを使用すると、`.extract()` は `attributePath` 引数で指定されたプロパティを持つコレクションを作成して返します。このコレクションのそれぞれの要素は、`targetPath` 引数のプロパティと、対応する`attributePath` 引数のプロパティを格納したオブジェクトです。Null値はそのまま保持されます（このシンタックスでは `option` に引数を渡しても無視されます）。

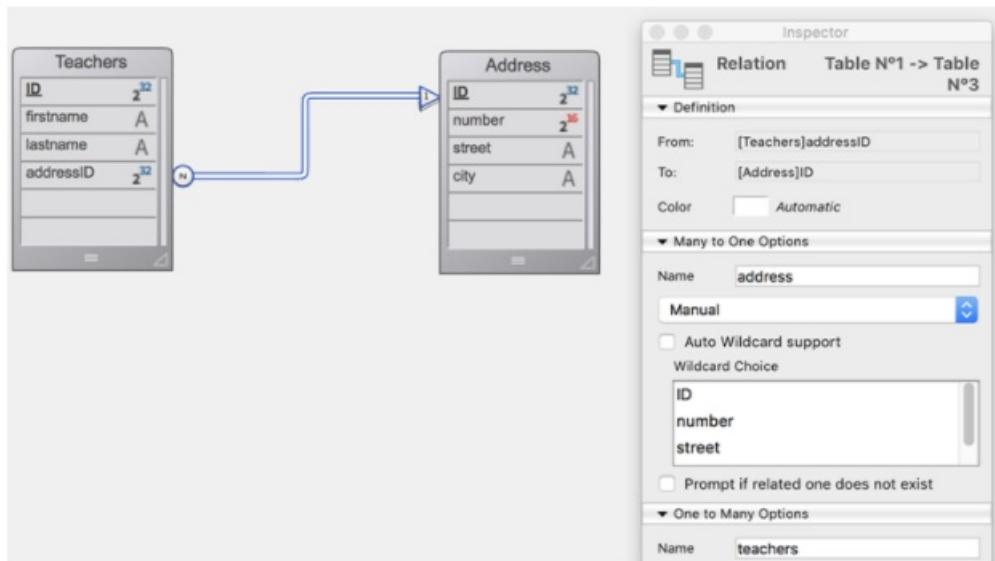
複数の `attributePath` 引数が渡した場合、それぞれに対して `targetPath` 引数を渡す必要があります。有効な `[attributePath, targetPath]` のペアのみが取得されます。

- `.kind = "relatedEntity"` であるデータクラス属性は、エンティティとして取得されます。
- `.kind = "relatedEntities"` であるデータクラス属性は、エンティティセレクションとして取得されます。

エンティティのコレクションにおいて、`[]` 記法を使用してアクセスしたエンティティは、データベースからは再読み込みされません。

例題

以下のテーブルとリレーションを前提とします：



```

var $firstnames; $addresses; $mailing; $teachers : Collection
//
//
// $firstnames は文字列のコレクション

$firstnames:=ds.Teachers.all().extract("firstname")
//
// $addresses は addressリレートエンティティのコレクション
// Null値も取得・保持されます
$addresses:=ds.Teachers.all().extract("address";ck keep null)
//
//
// $mailing は "who" および "to" のプロパティを持つオブジェクトのコレクション
// "who" プロパティの中身は文字列型
// "to" プロパティの中身はエンティティ型 (Address データクラス)
$mailing:=ds.Teachers.all().extract("lastname";"who";"address";"to")
//
//
// $mailing は "who" および "city" のプロパティを持つオブジェクトのコレクション
// "who" プロパティの中身は文字列型
// "city" プロパティの中身は文字列型
$mailing:=ds.Teachers.all().extract("lastname";"who";"address.city";"city")
//
// $teachers は"where" および "who" のプロパティを持つオブジェクトのコレクション
// "where" プロパティの中身は文字列型
// "who" プロパティの中身はエンティティセレクション (Teachers データクラス)
$teachers:=ds.Address.all().extract("city";"where";"teachers";"who")
//
// $teachers はエンティティセレクションのコレクション
$teachers:=ds.Address.all().extract("teachers")

```

.first()

▶ 覆歴

.first() : 4D.Entity

引数	タイプ		説明
戻り値	4D.Entity	<-	エンティティセレクションの先頭エンティティへの参照 (見つからなければ null)

説明

.first() 関数は、エンティティセレクションの先頭エンティティへの参照を返します。

この関数の結果は以下のコードに似ています:

```
$entity:=$entitySel[0]
```

ただし、この 2つの宣言には、セレクションが空であった場合に違いがあります:

```

var $entitySel : cs.EmpSelection
var $entity : cs.EmpEntity
$entitySel:=ds.Emp.query("lastName = :1";"Nonexistentname") // 合致するエンティティはありません
// エンティティセレクションは空になります
$entity:=$entitySel.first() // Null を返します
$entity:=$entitySel[0] // エラーを生成します

```

例題

```

var $entitySelection : cs.EmpSelection
var $entity : cs.EmpEntity
$entitySelection:=ds.Emp.query("salary > :1";100000)
If($entitySelection.length#0)
    $entity:=$entitySelection.first()
End if

```

.getDataClass()

▶ 覆歴

.getDataClass() : 4D.DataClass

引数	タイプ		説明
戻り値	4D.DataClass	<-	エンティティセレクションが所属しているデータクラス

説明

.getDataClass() 関数は、エンティティセレクションのデータクラスを返します。

このメソッドはおもに汎用的なコードのコンテキストで有用です。

例題

以下の汎用的なコードは、エンティティセレクションの全エンティティを複製します:

```

// duplicate_entities メソッド
// duplicate_entities($entity_selection)

#DECLARE ( $entitySelection : 4D.EntitySelection )
var $dataClass : 4D.DataClass
var $entity; $duplicate : 4D.Entity
var $status : Object
$dataClass:=$entitySelection.getDataClass()
For each($entity;$entitySelection)
    $duplicate:=$dataClass.new()
    $duplicate.fromObject($entity.toObject())
    $duplicate[$dataClass.getInfo().primaryKey]:=Null // プライマリーキーをリセットします
    $status:=$duplicate.save()
End for each

```

.getRemoteContextAttributes()

▶ 覆歴

.getRemoteContextAttributes() : Text

引数	タイプ		説明
result	テキスト	<-	エンティティセレクションにリンクされたコンテキスト属性 (カンマ区切り)

上級者向け: この機能は、特定の構成のため、ORDAのデフォルト機能をカスタマイズする必要がある開発者向けです。ほとんどの場合、使用する必要はないでしょう。

説明

.getRemoteContextAttributes() 関数は、エンティティセレクションによって使われている最適化コンテキストの情報を返します。

エンティティセレクションについて [最適化コンテキスト](#) が存在しない場合、関数は空のテキストを返します。

例題

```
var $ds : 4D.DataStoreImplementation
var $persons : cs.PersonsSelection
var $p : cs.PersonsEntity

var $info : Text
var $text : Text

$ds:=Open datastore(New object("hostname"; "www.myserver.com"); "myDS")

$persons:=$ds.Persons.all()
$text:=""
For each ($p; $persons)
    $text:=$p.firstname+" lives in "+$p.address.city+" / "
End for each

$info:=$persons.getRemoteContextAttributes()
//$info = "firstname,address,address.city"
```

参照

[Entity.getRemoteContextAttributes\(\)](#)
[.clearAllRemoteContexts\(\)](#)
[.getRemoteContextInfo\(\)](#)
[.getAllRemoteContexts\(\)](#)
[.setRemoteContextInfo\(\)](#)

.isAlterable()

▶履歴

.isAlterable() : Boolean

引数	タイプ		説明
戻り値	ブール	<-	エンティティセレクションが追加可能であれば true、それ以外の場合には false

説明

.isAlterable() 関数は、エンティティセレクションが追加可能の場合には true を返します。それ以外の場合には false を返します。

詳細については [共有可能/追加可能なエンティティセレクション](#) を参照ください。

例題

Form.products をフォーム内の [リストボックス](#) に表示し、ユーザーが新しい製品を追加できるようにします。ユーザーが製品を追加したときにエラーが起きないよう、追加可能であることを確認します：

```
If (Not(Form.products.isAlterable()))
    Form.products:=Form.products.copy()
End if
...
Form.products.add(Form.product)
```

.isOrdered()

▶履歴

.isOrdered() : Boolean

引数	タイプ		説明
戻り値	ブール	<-	順列ありエンティティセレクションの場合には true、そうでない場合は false

説明

.isOrdered() 関数は、エンティティセレクションが順列ありであれば true を返します。順列なしであれば false を返します。

リモートデータストアに属しているエンティティセレクションの場合は常に true を返します。

詳細については、[エンティティセレクションの順列あり/順列なし](#) を参照ください。

例題

```

var $employees : cs.EmployeeSelection
var $employee : cs.EmployeeEntity
var $isOrdered : Boolean
$employees:=ds.Employee.newSelection(dk keep ordered)
$employee:=ds.Employee.get(714) // プライマリーキー 714 を持つエンティティを取得します

// 順列ありのエンティティセレクションの場合、同じエンティティを複数回追加することができます(重複してるのは保持されます)
$employees.add($employee)
$employees.add($employee)
$employees.add($employee)

$isOrdered:=$employees.isOrdered()
If($isOrdered)
  ALERT("エンティティセレクションには順列があり、"+String($employees.length)+" 件の社員エンティティを含みます。")
End if

```

.last()

▶ 履歴

.last() : 4D.Entity

引数	タイプ		説明
戻り値	4D.Entity	<-	エンティティセレクションの最終エンティティへの参照 (見つからなければ null)

説明

.last() 関数は、エンティティセレクションの最終エンティティへの参照を返します。

この関数の結果は以下のコードに似ています:

```
$entity:=$entitySel[length-1]
```

エンティティセレクションが空の場合、関数は null を返します。

例題

```

var $entitySelection : cs.EmpSelection
var $entity : cs.EmpEntity
$entitySelection:=ds.Emp.query("salary < :1";50000)
If($entitySelection.length#0)
    $entity:=$entitySelection.last()
End if

```

.length

▶ 覆歴

.length : Integer

説明

.length プロパティは、エンティティセレクション内のエンティティの数を返します。エンティティセレクションが空の場合、関数は 0 を返します。

エンティティセレクションは、常に .length プロパティを持っています。

この関数は、元のエンティティセレクションを変更しません。

例題

```

var $vSize : Integer
$vSize:=ds.Employee.query("gender = :1";"male").length
ALERT(String(vSize)+" 人の男性社員が見つかりました。")

```

.max()

▶ 覆歴

.max(attributePath : Text) : any

引数	タイプ		説明
attributePath	テキスト	->	計算に使用する属性パス
戻り値	any	<-	属性の最大値

説明

.max() 関数は、attributePath に指定したエンティティセレクションの属性値のうち最高の（あるいは最大の）値を返します。実際にには、.orderBy() 関数を使用してエンティティセレクションを昇順に並べ替えたときの最後のエンティティを返します。

attributePath に、異なる型の値を格納しているオブジェクトプロパティを渡した場合、.max() メソッドは型のリスト順の中で最初のスカラー型の値の中の最大値を返します（.sort() の詳細を参照してください）。

エンティティセレクションが空の場合、または attributePath 引数がオブジェクト属性内に見つからない場合には、.max() は undefined を返します。

以下の場合には、エラーが返されます：

- attributePath はリレート属性である
- attributePath がエンティティセレクションデータクラス内に存在しない属性を指定している場合。

例題

女性従業員の中で最も高い給与額を探します：

```

var $sel : cs.EmpSelection
var $maxSalary : Real
$sel:=ds.Employee.query("gender = :1";"female")
$maxSalary:=$sel.max("salary")

```

.min()

▶履歴

.min(*attributePath* : Text) : any

引数	タイプ		説明
<i>attributePath</i>	テキスト	->	計算に使用する属性パス
戻り値	any	<-	属性の最小値

説明

.min() 関数は、*attributePath* に指定したエンティティセレクションの属性値のうち最低の（あるいは最小の）値を返します。実際に .orderBy() 関数を使用してエンティティセレクションを昇順に並べ替えたときの最初のエンティティを返します（null値は除く）。

attributePath に、異なる型の値を格納しているオブジェクトプロパティを渡した場合、.min() メソッドは型のリスト順の中で最初のスカラー型の値の中の最小値を返します（.sort() の詳細を参照してください）。

エンティティセレクションが空の場合、または *attributePath* 引数がオブジェクト属性内に見つからない場合には、.min() は undefined を返します。

以下の場合には、エラーが返されます：

- *attributePath* はリレート属性である
- *attributePath* がエンティティセレクションデータクラス内に存在しない属性を指定している場合。

例題

女性従業員の中で最も低い給与額を探します：

```

var $sel : cs.EmpSelection
var $minSalary : Real
$sel:=ds.Employee.query("gender = :1";"female")
$minSalary:=$sel.min("salary")

```

.minus()

▶履歴

.minus(*entity* : 4D.Entity) : 4D.EntitySelection

.minus(*entitySelection* : 4D.EntitySelection) : 4D.EntitySelection

引数	タイプ		説明
<i>entity</i>	4D.Entity	->	除外するエンティティ
<i>entitySelection</i>	4D.EntitySelection	->	除外するエンティティセレクション
戻り値	4D.EntitySelection	<-	新しいエンティティセレクション、あるいは既存のエンティティセレクションへの新しい参照

説明

.minus() 関数は、元のエンティティセレクションから、*entity* 引数のエンティティ、あるいは*entitySelection* 引数のエンティティセレクションに含まれるエンティティを除外し、結果のエンティティセレクションを返します。

- *entity* を引数として渡した場合、メソッドは (*entity* が元のエンティティセレクションに所属していた場合) *entity* を除外した新しいエンティティセレクションを作成します。*entity* が元のエンティティセレクションに含まれていなかった場合には、同エンティティセレクションへの新しい参照が返されます。
- *entitySelection* を引数として渡した場合、メソッドは *entitySelection* に所属しているエンティティを、元のエンティティセレクションから除外した新しいエンティティセレクションを返します。

順列ありと順列なしのエンティティセレクション を比較することができます。返されるセレクションは常に順列なしのものになります。

元のエンティティセレクションが空であった場合、空のエンティティセレクションが返されます。

entitySelection が空、あるいは *entity* が Null であった場合、元のエンティティセレクションへの新しい参照が返されます。

元のエンティティセレクションおよび引数が同じデータクラスのものでない場合、エラーが返されます。

例題 1

```
var $employees; $result : cs.EmployeeSelection
var $employee : cs.EmployeeEntity

$employees:=ds.Employee.query("lastName = :1";"H@")
// $employees エンティティセレクションは、主キー710 のエンティティと他のエンティティを格納しています
// 例: "Colin Hetrick", "Grady Harness", "Sherlock Holmes" (主キー710)

$employee:=ds.Employee.get(710) // "Sherlock Holmes" を返します

$result:=$employees.minus($employee) // $result には "Colin Hetrick", "Grady Harness" 格納されます
```

例題 2

"Jones" という名前で、New York に住んでいる女性従業員のセレクションを取得します:

```
var $sel1; $sel2; $sel3 : cs.EmployeeSelection
$sel1:=ds.Employee.query("name =:1";"Jones")
$sel2:=ds.Employee.query("city=:1";"New York")
$sel3:=$sel1.and($sel2).minus(ds.Employee.query("gender='male'"))
```

.or()

▶ 履歴

```
.or( entity : 4D.Entity ) : 4D.EntitySelection
.or( entitySelection : 4D.EntitySelection ) : 4D.EntitySelection
```

引数	タイプ		説明
entity	4D.Entity	->	交差するエンティティ
entitySelection	4D.EntitySelection	->	交差するエンティティセレクション
戻り値	4D.EntitySelection	<-	新しいエンティティセレクション、あるいは元のエンティティセレクションへの新しい参照

説明

.or() 関数は、OR論理演算子を使用して、*entity* または *entitySelection* のエンティティと対象エンティティセレクションを組み合わせます。戻り値は、渡した引数とエンティティセレクションの全エンティティを格納する順列なしの新規エンティティセレクションです。

- *entity* を渡した場合、引数のエンティティをエンティティセレクションと比較することになります。エンティティがエンティティセレクションに所属している場合、エンティティセレクションへの新しい参照が返されます。そうでない場合、元のエンティティセレクションと渡したエンティティを格納した新しいエンティティセレクションが返されます。
- *entitySelection* を渡した場合、二つのエンティティセレクションを比較することになります。元のエンティティセレクションと *entitySelection* のどち

らかに所属しているエンティティを格納した新しいエンティティセレクションが返されます (OR は排他的ではなく、また両方のセレクションで参照されているエンティティは、結果のセレクションに複数格納されることはありません)。

順列ありと順列なしのエンティティセレクション を比較することができます。返されるセレクションは常に順列なしのものになります。

元のエンティティセレクションと `entitySelection` の両方が空であった場合、空のエンティティセレクションが返されます。元のエンティティセレクションが空であった場合、`entitySelection` への参照、あるいは `entity` のみを格納したエンティティセレクションが返されます。

`entitySelection` が空、あるいは `entity` が Null であった場合、元のエンティティセレクションへの新しい参照が返されます。

元のエンティティセレクションおよび引数が同じデータクラスのものでない場合、エラーが返されます。

例題 1

```
var $employees1; $employees2; $result : cs.EmployeeSelection
$employees1:=ds.Employee.query("lastName = :1";"H@") // "Colin Hetrick", "Grady Harness" を返します
$employees2:=ds.Employee.query("firstName = :1";"C@") // "Colin Hetrick", "Cath Kidston" を返します
$result:=$employees1.or($employees2) // $result には "Colin Hetrick", "Grady Harness", "Cath Kidston" が格納されます
```

例題 2

```
var $employees; $result : cs.EmployeeSelection
var $employee : cs.EmployeeEntity
$employees:=ds.Employee.query("lastName = :1";"H@") // "Colin Hetrick", "Grady Harness", "Sherlock Holmes" を返します
$employee:=ds.Employee.get(686) // 主キー686 のエンティティは $employees エンティティセレクションに属していません
// 主キー686 は "Mary Smith" という従業員に合致します

$result:=$employees.or($employee) // $result には "Colin Hetrick", "Grady Harness", "Sherlock Holmes", "Mary Smith" が格納されます
```

.orderBy()

▶履歴

.orderBy(`pathString` : Text) : 4D.EntitySelection
.orderBy(`pathObjects` : Collection) : 4D.EntitySelection

引数	タイプ		説明
<code>pathString</code>	テキスト	->	エンティティセレクションの属性パスと並べ替えの指定
<code>pathObjects</code>	コレクション	->	条件オブジェクトのコレクション
戻り値	4D.EntitySelection	<-	指定された順番に並べ替えられた新規エンティティセレクション

説明

`.orderBy()` 関数は、エンティティセレクションの全エンティティが `pathString` または `pathObjects` が指定する順番に並べ替えられた、新規の順列ありのエンティティセレクションを返します。

- この関数は、エンティティセレクションを変更しません。
 - 詳細については、[エンティティセレクションの順列あり/順列なし](#) を参照ください。
- 引数を渡して、エンティティの並び替えを指定する必要があります。並べ替えの指定方法は 2つあります:
- `pathString` (テキスト): この場合、カンマ区切りされた、1 ~ x 個の属性パスと並べ替え順 (任意) で構成されるフォーミュラを渡します。シンタックスは以下の通りです:

```
"attributePath1 {desc or asc}, attributePath2 {desc or asc},..."
```

属性を渡す順番が、エンティティの並べ替えの優先順位を決定します。デフォルトでは、属性は昇順に並べ替えられます。並び順を設定するには、プロパティパスの後に半角スペースで区切ったあとに、昇順を指定するには "asc"、降順を指定するには "desc" を渡します。

- *pathObjects* (コレクション): コレクションの各要素は、以下の構造を持つオブジェクトを格納します:

```
{  
    "propertyPath": string,  
    "descending": boolean  
}
```

デフォルトでは、属性は昇順に並べ替えられます ("descending" は false)。

pathObjects コレクションには必要な数だけオブジェクトを追加することができます。

Null は他の値より小さいと評価されます。

例題

```
// フォーミュラでの並べ替え  
$sortedEntitySelection:=$entitySelection.orderBy("firstName asc, salary desc")  
$sortedEntitySelection:=$entitySelection.orderBy("firstName")  
  
// コレクションでの並べ替えと、昇順・降順の指定  
$orderColl:=New collection  
$orderColl.push(New object("propertyPath";"firstName";"descending";False))  
$orderColl.push(New object("propertyPath";"salary";"descending";True))  
$sortedEntitySelection:=$entitySelection.orderBy($orderColl)  
  
$orderColl:=New collection  
$orderColl.push(New object("propertyPath";"manager.lastName"))  
$orderColl.push(New object("propertyPath";"salary"))  
$sortedEntitySelection:=$entitySelection.orderBy($orderColl)
```

.orderByFormula()

▶履歴

.orderByFormula(*formulaString* : Text { ; *sortOrder* : Integer } { ; *settings* : Object}) : 4D.EntitySelection
.orderByFormula(*formulaObj* : Object { ; *sortOrder* : Integer } { ; *settings* : Object}) : 4D.EntitySelection

引数	タイプ		説明
<i>formulaString</i>	テキスト	->	フォーミュラ文字列
<i>formulaObj</i>	オブジェクト	->	フォーミュラオブジェクト
<i>sortOrder</i>	整数	->	dk ascending (デフォルト) または dk descending
<i>settings</i>	オブジェクト	->	フォーミュラに渡す引数
戻り値	4D.EntitySelection	<-	順列ありの新規エンティティセレクション

説明

.orderByFormula() 関数は、エンティティセレクションの全エンティティが *formulaString* または *formulaObj*、および (任意の) *sortOrder* や *settings* 引数が指定する順番に並べられた、順列ありの新規エンティティセレクションを返します。

この関数は、元のエンティティセレクションを変更しません。

formulaString または *formulaObj* 引数を渡すことができます:

- *formulaString*: "Year of(this.birthDate)" などの 4D式
- *formulaObj*: `Formula` または `Formula from string` コマンドを使用して作成された、有効なフォーミュラオブジェクト

formulaString および *formulaObj* はエンティティセレクションの各エンティティに対して実行され、その結果は返されるエンティティセレクション内でのエンティティの位置を決定するのに使用されます。結果は並べ替え可能な型 (布尔、日付、数値、テキスト、時間、Null) である必要があります。

Null値の結果は常に最小の値とみなされます。

sortOrder 引数を省略した場合のデフォルトでは、返されるエンティティセレクションは昇順に並べられます。オプションとして、*sortOrder* に以下の値のいずれか一つを渡すことができます:

定数	値	説明
dk ascending	0	昇順 (デフォルト)
dk descending	1	降順

formulaString および *formulaObj* 内では、処理されるエンティティとその属性は `This` コマンドを通して利用可能です (たとえば、`This.lastName` など)。

`settings` 引数の `args` プロパティ (オブジェクト) を使用することで、フォーミュラに引数を渡すことが可能です。このときフォーミュラは、`settings.args` オブジェクトを \$1 に受け取ります。

例題 1

テキストとして渡されたフォーミュラを使用して学生を並べ替えます:

```
var $es1; $es2 : cs.StudentsSelection
$es1:=ds.Students.query("nationality=:1";"French")
$es2:=$es1.orderByFormula("length(this.lastname)") // デフォルトで昇順
$es2:=$es1.orderByFormula("length(this.lastname)";dk descending)
```

同じ並び方を、フォーミュラオブジェクトを使用して指定します:

```
var $es1; $es2 : cs.StudentsSelection
var $formula : Object
$es1:=ds.Students.query("nationality=:1";"French")
$formula:=Formula(Length(This.lastname))
$es2:=$es1.orderByFormula($formula) // デフォルトで昇順
$es2:=$es1.orderByFormula($formula;dk descending)
```

例題 2

引数付きのフォーミュラオブジェクトを渡します。`settings.args` オブジェクトは、`computeAverage` メソッド内で \$1 が受け取ります。

この例題では、`Students` データクラス内の "marks" オブジェクトフィールドに科目ごとの生徒の成績が格納されています。フォーミュラオブジェクトを使用し、`schoolA` と `schoolB` で異なる係数を用いて生徒の平均の成績を計算します。

```

var $es1; $es2 : cs.StudentsSelection
var $formula; $schoolA; $schoolB : Object
$es1:=ds.Students.query("nationality=:1;" "French")
$formula:=Formula(computeAverage($1))

$schoolA:=New object() // settingsオブジェクト
$schoolA.args:=New object("english";1;"math";1;"history";1) // 平均を計算するための係数

// school A の条件に応じて学生を並べ替えます
$es2:=$es1.entitySelection.orderByFormula($formula;$schoolA)

$schoolB:=New object() // settingsオブジェクト
$schoolB.args:=New object("english";1;"math";2;"history";3) // 平均を計算するための係数

// school B の条件に応じて学生を並べ替えます
$es2:=$es1.entitySelection.orderByFormula($formula;dk descending;$schoolB)

```

```

//
// computeAverage メソッド
// -----
#DECLARE ($coefList : Object) -> $result : Integer
var $subject : Text
var $average; $sum : Integer

$average:=0
$sum:=0

For each($subject;$coefList)
    $sum:=$sum+$coefList[$subject]
End for each

For each($subject;This.marks)
    $average:=$average+(This.marks[$subject]*$coefList[$subject])
End for each

$result:=$average/$sum

```

.query()

▶ 履歴

.query(*queryString* : Text { ; ...*value* : any } { ; *querySettings* : Object }) : 4D.EntitySelection
 .query(*formula* : Object { ; *querySettings* : Object }) : 4D.EntitySelection

引数	タイプ		説明
<i>queryString</i>	テキスト	->	検索条件 (文字列)
<i>formula</i>	オブジェクト	->	検索条件 (フォーミュラオブジェクト)
<i>value</i>	any	->	プレースホルダー用の値
<i>querySettings</i>	オブジェクト	->	クエリオプション: parameters, attributes, args, allowFormulas, context, queryPath, queryPlan
戻り値	4D.EntitySelection	<-	<i>queryString</i> または <i>formula</i> に渡した検索条件に合致する、エンティティセレクション内のエンティティから構成された新しいエンティティセレクション

説明

.query() 関数は、エンティティセレクションの全エンティティから、*queryString* または *formula* と任意の *value* 引数で指定した検索条件に合

致するエンティティを検索します。戻り値は、見つかったエンティティをすべて格納する `EntitySelection` 型の新しいオブジェクトです。この関数には、レイジーローディングが適用されます。

この関数は、元のエンティティセレクションを変更しません。

エンティティが見つからない場合、空のエンティティセレクションが返されます。

`queryString` および `value` や `querySettings` パラメーターを使ってクエリをビルドする方法の詳細については、`DataClass .query()` 関数を参照ください。

`queryString` 内で `order by` ステートメントを省略した場合のデフォルトでは、返されるエンティティセレクションは、順列なしのものになります。しかしながら、クライアント/サーバーモードにおいては、順列ありのエンティティセレクションのように振る舞う（エンティティはセレクションの終わりに追加されていく）点に注意してください。

例題 1

```
var $entitySelectionTemp : cs.EmployeeSelection
$entitySelectionTemp:=ds.Employee.query("lastName = :1";"M@")
Form.emps:=$entitySelectionTemp.query("manager.lastName = :1";"S@")
```

例題 2

追加のクエリ例については、`DataClass.query()` を参照してください。

参照

データクラスの `.query()`

.queryPath

▶ 履歴

`.queryPath` : Text

説明

`.queryPath` プロパティは、実際に 4Dで実行されたクエリの詳細な情報を格納します。このプロパティは、`.query()` 関数の `querySettings` 引数に `"queryPath":true` プロパティが渡されていた場合に、クエリを通して生成された `EntitySelection` オブジェクトで利用可能です。

詳細については、`DataClass .query()` の `querySettings` の説明を参照ください。

.queryPlan

▶ 履歴

`.queryPlan` : Text

説明

`.queryPlan` プロパティは、実行前のクエリの詳細な情報（クエリプラン）を格納します。このプロパティは、`.query()` 関数の `querySettings` 引数に `"queryPlan":true` プロパティが渡されていた場合に、クエリを通して生成された `EntitySelection` オブジェクトで利用可能です。

詳細については、`DataClass .query()` の `querySettings` の説明を参照ください。

.refresh()

▶ 履歴

`.refresh()`

| 引数 | タイプ | | 説明 | | -- | --- | ::| ----- | | | | このコマンドは引数を必要としません |

説明

このメソッドはリモートデータストア（クライアント/サーバーモード、または Open datastore 接続）においてのみ動作します。

.refresh() 関数は、ローカルの ORDA キャッシュにあるエンティティセレクションデータを即座に "無効化" します。そのため、次に 4D がエンティティセレクションを必要としたときにはそれがデータベースからリロードされます。

デフォルトでは、ローカルの ORDA のキャッシュは 30 秒後に無効化されます。クライアント/サーバーアプリケーションのコンテキストにおいて ORDA とクラシック言語の両方を使用している場合、このメソッドを使用することでリモートアプリケーションが必ず最新のデータを使用するようにできます。

例題 1

クラシックと ORDA 言語の両方が、同じデータを同時に編集する場合を考えます：

```
// 4Dリモート上で実行

var $selection : cs.StudentsSelection
var $student : cs.StudentsEntity

$selection:=ds.Students.query("lastname=:1";"Collins")
// 先頭エンティティを ORDA のキャッシュに読み込みます
$student:=$selection.first()

// 4Dのクラシック言語でデータを更新します。ORDA キャッシュはこれを検知しません
QUERY([Students];[Students]lastname="Collins")
[Students]lastname:="Colin"
SAVE RECORD([Students])

// 最新情報を取得するため、ORDA キャッシュを無効化する必要があります
$selection.refresh()
// キャッシュが失効していないなくても、先頭エンティティがディスクから再読み込みされます
$student:=$selection.first()

// $student.lastname contains "Colin"
```

例題 2

リストボックスに Form.students エンティティセレクションを表示し、複数のクライアントがそれを操作する場合を考えます。

```
// フォームメソッド：
Case of
:(Form event code=On Load)
  Form.students:=ds.Students.all()
End case
//
//
// クライアント#1で、ユーザーが先頭エンティティをロードし、更新して保存します
// クライアント#2でも、ユーザーが同じエンティティをロードし、更新して保存します
//
//
// クライアント#1において：
Form.students.refresh() // Form.students エンティティセレクションの ORDA キャッシュを無効化します
// リストボックスの中身はデータベースの内容で更新され、クライアント#2 がおこなった変更も反映します
```

.selected()

▶ 覆歴

`.selected(selectedEntities : 4D.EntitySelection) : Object`

引数	タイプ		説明
selectedEntities	4D.EntitySelection	->	呼び出し対象のエンティティセレクションにおける、選別したエンティティの位置範囲
戻り値	オブジェクト	<-	呼び出し対象のエンティティセレクション内での位置を取得したい、選別されたエンティティのセレクション

説明

呼び出し対象のエンティティセレクション、または `selectedEntities` のエンティティセレクションが空の場合、関数は `ranges` に空のコレクションを返します。

この関数は、元のエンティティセレクションを変更しません。

`entity` を引数として渡した場合、メソッドは (`entity` が元のエンティティセレクションに所属していた場合) `entity` を除外した新しいエンティティセレクションを作成します。`entity` が元のエンティティセレクションに含まれていなかった場合には、同エンティティセレクションへの新しい参照が返されます。

戻り値

戻り値のオブジェクトには、以下のプロパティが格納されています:

プロパティ	タイプ	説明
<code>ranges</code>	Collection	レンジオブジェクトのコレクション
<code>ranges[]</code> .start	Integer	レンジ内の先頭エンティティのインデックス (位置)
<code>ranges[]</code> .end	Integer	レンジ内の最終エンティティのインデックス (位置)

`ranges` プロパティに 1件のエンティティしか含まれない場合、`start` = `end` です。インデックスは 0 起点です。

`attributeName` で指定した属性がストレージ型の場合: `.attributeName` は `attributeName` と同じ型の値のコレクションを返します。

例題

```
var $invoices; $cashSel; $creditSel : cs.Invoices
var $result1; $result2 : Object

$invoices:=ds.Invoices.all()

$cashSelection:=ds.Invoices.query("payment = :1"; "Cash")
$creditSel:=ds.Invoices.query("payment IN :1"; New collection("Cash"; "Credit Card"))

$result1:=$invoices.selected($cashSelection)
$result2:=$invoices.selected($creditSel)

//$/result1 = {ranges: [{start:0;end:0},{start:3;end:3},{start:6;end:6}]}
//$/result2 = {ranges: [{start:0;end:1},{start:3;end:4},{start:6;end:7}]}
```

.slice()

▶ 履歴

`.slice(startFrom : Integer { ; end : Integer }) : 4D.EntitySelection`

引数	タイプ		説明
startFrom	整数	->	処理を開始するインデックス)
end	整数	->	終了インデックス (含まれない)
戻り値	4D.EntitySelection	<-	抜粋エンティティを格納した新しいエンティティセレクション (シャロウ・コピー)

説明

.slice() 関数は、startFrom の位置 (含まれる) から end の位置 (含まれない) または終わりまでのエンティティセレクションの一部を、新規エンティティセレクションとして返します。この関数は、エンティティセレクションのシャロウ・コピーを返します (同じエンティティ参照を使用します)。

この関数は、元のエンティティセレクションを変更しません。

戻り値のエンティティセレクションには、startFrom 引数で指定したエンティティ (含まれる) から、end 引数で指定したエンティティまで (含まれない) の全エンティティが格納されます。startFrom 引数のみを渡した場合には、startFrom 引数で指定したエンティティから最後のエンティティまでが戻り値のエンティティセレクションに格納されます。

- startFrom < 0 の場合、startFrom:=startFrom+length として再計算されます (エンティティセレクションの終端からのオフセットであるとみなされます)。再計算された値も負の値だった場合、startFrom は 0 に設定されます。
- startFrom >= length の場合、関数は空のエンティティセレクションを返します。
- end < 0 の場合、それは end:=end+length として再計算されます。
- 渡された値、あるいは再計算された値が end < startFrom の場合、関数はなにもしません。

エンティティセレクションにドロップされたエンティティが含まれる場合、それらも返されます。

例題 1

エンティティセレクションの、最初の 9 件のエンティティのセレクションを取得します:

```
var $sel; $sliced : cs.EmployeeSelection
$sel:=ds.Employee.query("salary > :1";50000)
$sliced:=$sel.slice(0;9) //
```

例題 2

ds.Employee.all().length = 10 である場合:

```
var $slice : cs.EmployeeSelection
$slice:=ds.Employee.all().slice(-1;-2) // インデックス 9 から 8 番までを返そうとしますが、9 > 8 なので空のセレクション
```

.sum()

▶履歴

.sum(attributePath : Text) : Real

引数	タイプ		説明
attributePath	テキスト	->	計算に使用する属性パス
戻り値	実数	<-	エンティティセレクションの値の合計

説明

.sum() 関数は、attributePath に指定したエンティティセレクションの属性値の総和を返します。

エンティティセレクションが空の場合、`.sum()` は 0 を返します。

総和は、数値型の値に対してのみ実行可能です。`attributePath` がオブジェクトプロパティだった場合、計算の対象になるのは数値型の値のみです（他の値の型は無視されます）。この場合で、`attributePath` がオブジェクト内に存在しないパス、あるいは数値を含んでいない属性へのパスであった場合には、`.sum()` は 0 を返します。

以下の場合には、エラーが返されます：

- `attributePath` が数値型あるいはオブジェクト型の属性ではない
- `attributePath` はリレート属性である
- `attributePath` がエンティティセレクションデータクラス内に存在しない場合。

例題

```
var $sel : cs.EmployeeSelection
var $sum : Real

$sel:=ds.Employee.query("salary < :1";20000)
$sum:=$sel.sum("salary")
```

.toCollection()

▶履歴

`.toCollection({ options : Integer { ; begin : Integer { ; howMany : Integer } } }) : Collection`
`.toCollection(filterString : Text {; options : Integer { ; begin : Integer { ; howMany : Integer } } }) : Collection`
`.toCollection(filterCol : Collection {; options : Integer { ; begin : Integer { ; howMany : Integer } } }) : Collection`

引数	タイプ		説明
filterString	テキスト	->	抽出するエンティティの属性パスの文字列
filterCol	コレクション	->	抽出するエンティティの属性パスのコレクション
options	整数	->	<code>dk with primary key</code> : プライマリーキーを追加 <code>dk with stamp</code> : スタンプを追加
begin	整数	->	開始インデックス
howMany	整数	->	抽出するエンティティ数
戻り値	コレクション	<-	エンティティセレクションの属性と値を格納したオブジェクトのコレクション

説明

`.toCollection()` 関数は、エンティティセレクションの各エンティティの属性名と値に対応するプロパティと値のセットを持つオブジェクト要素を格納するコレクションを作成し、返します。

`filterString` および `filterCol` 引数が省略されるか、空の文字列が渡されるか、あるいは "*" が渡された場合、すべての属性が抽出されます。`"kind"` プロパティが "relatedEntity" の属性は単純な形式で抽出されます: __KEY プロパティ (プライマリーキー) を持ったオブジェクト。`"relatedEntities"` 型の "kind" プロパティの属性は抽出されません。

抽出するエンティティ属性を限定したい場合には、それを指定する引数を渡すことができます。2つのシンタックスを使用できます:

- `filterString`: プロパティパスをカンマで区切った文字列: "propertyPath1, propertyPath2, ..."
- `filterCol`: プロパティパスを含む文字列のコレクション: ["propertyPath1", "propertyPath2", ...]

引数が、`relatedEntity` (リレートエンティティ) 型の属性を指定していた場合:

- `propertyPath = "relatedEntity"` -> 単純な形式で取得されます
- `propertyPath = "relatedEntity.*"` -> 全プロパティが取得されます
- `propertyPath = "relatedEntity.propertyName1, relatedEntity.propertyName2, ..."` -> 指定されたプロパティのみが取得されます

引数が、`relatedEntities` (リレートエンティティズ) 型の属性を指定していた場合:

- `propertyPath = "relatedEntities.*"` -> 全プロパティが取得されます
- `propertyPath = "relatedEntities.propertyName1, relatedEntities.propertyName2, ..."` -> 指定されたプロパティのみが取得されます

`options` に `dk with primary key` または `dk with stamp` セレクターを渡すことで、エンティティのプライマリーキー/スタンプを、取得するオブジェクトに追加するかどうかを指定できます。

`begin` 引数を渡すことで、抽出するエンティティの開始インデックスを指定することができます。0 からエンティティセレクションの長さ - 1 の範囲で値を渡すことができます。

`howMany` 引数を渡すと、`begin` 引数で指定した位置から抽出するエンティティの件数を指定することができます。ドロップされたエンティティは返されませんが、`howMany` 引数のカウントでは考慮されます。たとえば、ドロップされたエンティティが1つある場合に `howMany = 3` であれば、2件のエンティティが抽出されます。

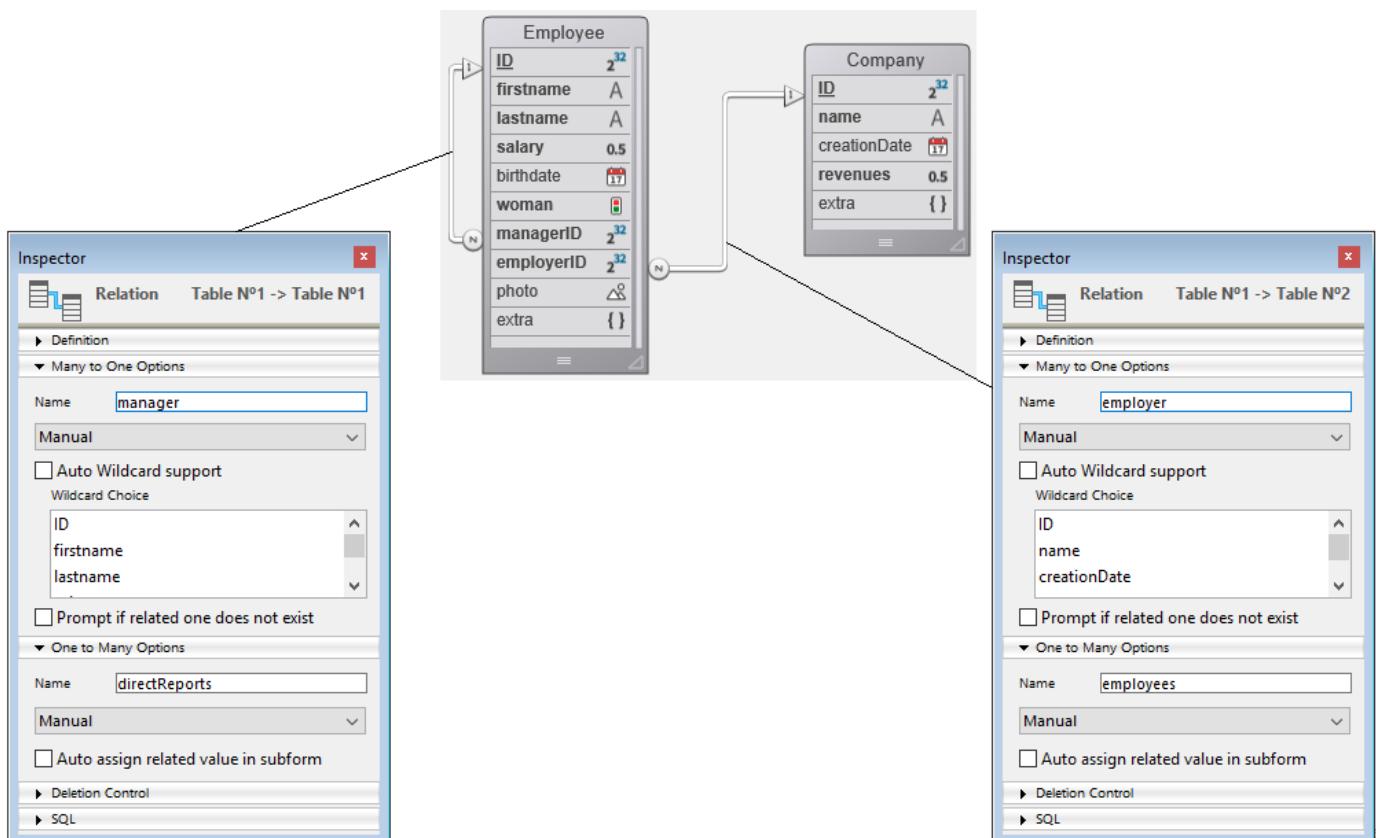
`howMany` 引数がエンティティセレクションの `length` を超える場合、メソッドは(`length - begin`) の数だけオブジェクトを返します。

以下のいずれかの場合には空のコレクションが返されます:

- エンティティセレクションが空である
- `begin` 引数がエンティティセレクションの `length` を超えている

例題 1

このセクションの例題では、以下のストラクチャーを使います:



`filterString` や `filterCol`、および `options` 引数を渡さない例:

```

var $employeesCollection : Collection
var $employees : cs.EmployeeSelection

$employeesCollection:=New collection
$employees:=ds.Employee.all()
$employeesCollection:=$employees.toCollection()

```

戻り値:

```
[
  {
    "ID": 416,
    "firstName": "Gregg",
    "lastName": "Wahl",
    "salary": 79100,
    "birthDate": "1963-02-01T00:00:00.000Z",
    "woman": false,
    "managerID": 412,
    "employerID": 20,
    "photo": "[object Picture]",
    "extra": null,
    "employer": {
      "__KEY": 20
    },
    "manager": {
      "__KEY": 412
    }
  },
  {
    "ID": 417,
    "firstName": "Irma",
    "lastName": "Durham",
    "salary": 47000,
    "birthDate": "1992-06-16T00:00:00.000Z",
    "woman": true,
    "managerID": 412,
    "employerID": 20,
    "photo": "[object Picture]",
    "extra": null,
    "employer": {
      "__KEY": 20
    },
    "manager": {
      "__KEY": 412
    }
  }
]
]
```

例題 2

options を使用した例:

```
var $employeesCollection : Collection
var $employees : cs.EmployeeSelection

$employeesCollection:=New collection
$employees:=ds.Employee.all()
$employeesCollection:=$employees.toCollection("");dk with primary key+dk with stamp)
```

戻り値:

```
[
  {
    "__KEY": 416,
    "__STAMP": 1,
    "ID": 416,
    "firstName": "Gregg",
    "lastName": "Wahl",
    "salary": 79100,
    "birthDate": "1963-02-01T00:00:00.000Z",
    "woman": false,
    "managerID": 412,
    "employerID": 20,
    "photo": "[object Picture]",
    "extra": null,
    "employer": {
      "__KEY": 20
    },
    "manager": {
      "__KEY": 412
    }
  },
  {
    "__KEY": 417,
    "__STAMP": 1,
    "ID": 417,
    "firstName": "Irma",
    "lastName": "Durham",
    "salary": 47000,
    "birthDate": "1992-06-16T00:00:00.000Z",
    "woman": true,
    "managerID": 412,
    "employerID": 20,
    "photo": "[object Picture]",
    "extra": null,
    "employer": {
      "__KEY": 20
    },
    "manager": {
      "__KEY": 412
    }
  }
]
```

例題 3

抽出件数と、filterCol で抽出プロパティを指定した例:

```
var $employeesCollection; $filter : Collection
var $employees : cs.EmployeeSelection

$employeesCollection:=New collection
$filter:=New collection

$filter.push("firstName")
$filter.push("lastName")

$employees:=ds.Employee.all()
$employeesCollection:=$employees.toCollection($filter;0;0;2)
```

戻り値:

```
[  
  {  
    "firstName": "Gregg",  
    "lastName": "Wahl"  
  },  
  {  
    "firstName": "Irma",  
    "lastName": "Durham"  
  }  
]
```

例題 4

`relatedEntity` (リレートエンティティ) 型の属性を単純な形式で抽出した例:

```
var $employeesCollection : Collection  
$employeesCollection:=New collection  
$employeesCollection:=$employees.toCollection("firstName,lastName,employer")
```

戻り値:

```
[  
  {  
    "firstName": "Gregg",  
    "lastName": "Wahl",  
    "employer": {  
      "__KEY": 20  
    }  
  },  
  {  
    "firstName": "Irma",  
    "lastName": "Durham",  
    "employer": {  
      "__KEY": 20  
    }  
  },  
  {  
    "firstName": "Lorena",  
    "lastName": "Boothe",  
    "employer": {  
      "__KEY": 20  
    }  
  }  
]
```

例題 5

`filterCol` を使用した例:

```
var $employeesCollection; $coll : Collection  
$employeesCollection:=New collection  
$coll:=New collection("firstName";"lastName")  
$employeesCollection:=$employees.toCollection($coll)
```

戻り値:

```
[
  {
    "firstName": "Joanna",
    "lastName": "Cabrera"
  },
  {
    "firstName": "Alexandra",
    "lastName": "Coleman"
  }
]
```

例題 6

リレートエンティティの全プロパティを抽出する例:

```
var $employeesCollection; $coll : Collection
$employeesCollection:=New collection
$coll:=New collection
$coll.push("firstName")
$coll.push("lastName")
$coll.push("employer.*")
$employeesCollection:=$employees.toCollection($coll)
```

戻り値:

```
[
  {
    "firstName": "Gregg",
    "lastName": "Wahl",
    "employer": {
      "ID": 20,
      "name": "India Astral Secretary",
      "creationDate": "1984-08-25T00:00:00.000Z",
      "revenues": 12000000,
      "extra": null
    }
  },
  {
    "firstName": "Irma",
    "lastName": "Durham",
    "employer": {
      "ID": 20,
      "name": "India Astral Secretary",
      "creationDate": "1984-08-25T00:00:00.000Z",
      "revenues": 12000000,
      "extra": null
    }
  },
  {
    "firstName": "Lorena",
    "lastName": "Boothe",
    "employer": {
      "ID": 20,
      "name": "India Astral Secretary",
      "creationDate": "1984-08-25T00:00:00.000Z",
      "revenues": 12000000,
      "extra": null
    }
  }
]
```

例題 7

リレートエンティティの一部のプロパティを抽出する例:

```
var $employeesCollection : Collection
$employeesCollection:=New collection
$employeesCollection:=$employees.toCollection("firstName, lastName, employer.name")
```

```
[  
  {  
    "firstName": "Gregg",  
    "lastName": "Wahl",  
  
    "employer": {  
      "name": "India Astral Secretary"  
    }  
  },  
  {  
    "firstName": "Irma",  
    "lastName": "Durham",  
    "employer": {  
      "name": "India Astral Secretary"  
    }  
  },  
  {  
    "firstName": "Lorena",  
    "lastName": "Boothe",  
    "employer": {  
      "name": "India Astral Secretary"  
    }  
  }]  
}]
```

例題 8

`relatedEntities` (リレートエンティティズ) の一部のプロパティを抽出する例:

```
var $employeesCollection : Collection
$employeesCollection:=New collection
$employeesCollection:=$employees.toCollection("firstName, lastName, directReports.firstName")
```

戻り値:

```
[
  {
    "firstName": "Gregg",
    "lastName": "Wahl",
    "directReports": []
  },
  {
    "firstName": "Mike",
    "lastName": "Phan",
    "directReports": [
      {
        "firstName": "Gary"
      },
      {
        "firstName": "Sadie"
      },
      {
        "firstName": "Christie"
      }
    ]
  },
  {
    "firstName": "Gary",
    "lastName": "Reichert",
    "directReports": [
      {
        "firstName": "Rex"
      },
      {
        "firstName": "Jenny"
      },
      {
        "firstName": "Lowell"
      }
    ]
  }
]
```

例題 9

`relatedEntities` (リレートエンティティズ) の全プロパティを抽出する例:

```
var $employeesCollection : Collection
$employeesCollection:=New collection
$employeesCollection:=$employees.toCollection("firstName, lastName, directReports.*")
```

```
[
  {
    "firstName": "Gregg",
    "lastName": "Wahl",
    "directReports": []
  },
  {
    "firstName": "Mike",
    "lastName": "Phan",
    "directReports": [
      {
        "ID": 425,
        "firstName": "Gary",
        "lastName": "Reichert",
        "directReports": []
      }
    ]
  }
]
```

```
        "salary": 65800,
        "birthDate": "1957-12-23T00:00:00.000Z",
        "woman": false,
        "managerID": 424,
        "employerID": 21,
        "photo": "[object Picture]",
        "extra": null,
        "employer": {
            "__KEY": 21
        },
        "manager": {
            "__KEY": 424
        }
    },
    {
        "ID": 426,
        "firstName": "Sadie",
        "lastName": "Gallant",
        "salary": 35200,
        "birthDate": "2022-01-03T00:00:00.000Z",
        "woman": true,
        "managerID": 424,
        "employerID": 21,
        "photo": "[object Picture]",
        "extra": null,
        "employer": {
            "__KEY": 21
        },
        "manager": {
            "__KEY": 424
        }
    }
],
},
{
    "firstName": "Gary",
    "lastName": "Reichert",
    "directReports": [
        {
            "ID": 428,
            "firstName": "Rex",
            "lastName": "Chance",
            "salary": 71600,
            "birthDate": "1968-08-09T00:00:00.000Z",
            "woman": false,

            "managerID": 425,
            "employerID": 21,
            "photo": "[object Picture]",
            "extra": null,
            "employer": {
                "__KEY": 21
            },
            "manager": {
                "__KEY": 425
            }
        },
        {
            "ID": 429,
            "firstName": "Jenny",
            "lastName": "Parks",
            "salary": 51300,
            "birthDate": "1984-05-25T00:00:00.000Z",
            "woman": true,
            "managerID": 425,
            "extra": null
        }
    ]
}
]
```

```
"employerID": 21,  
"photo": "[object Picture]",  
"extra": null,  
"employer": {  
    "__KEY": 21  
},  
"manager": {  
    "__KEY": 425  
}  
}  
]  
}  
]
```

File

`File` オブジェクトは `File` コマンドによって作成されます。これらのオブジェクトには、(実在しているか否かに関わらず) ディスクファイルへの参照が格納されます。たとえば、新規ファイルを作成するために `File` コマンドを実行した場合、有効な `File` オブジェクトが作成されます
が、`file.create()` 関数を呼び出すまで、ディスク上にはなにも保存されていません。

例題

プロジェクトフォルダーにプリファレンスファイルを作成します:

```
var $created : Boolean  
$created:=File("/PACKAGE/SpecialPrefs/"+Current user+".myPrefs").create()
```

File オブジェクト

`.copyTo(destinationFolder : 4D.Folder { ; newName : Text } { ; overwrite : Integer }) : 4D.File`

`File` オブジェクトを、`destinationFolder` 引数で指定したフォルダーへとコピーします

`.create() : Boolean`

`File` オブジェクトのプロパティに基づいてディスク上にファイルを作成します

`.createAlias(destinationFolder : 4D.Folder ; aliasName : Text { ; aliasType : Integer }) : 4D.File`

エイリアス (macOS) またはショートカット (Windows) を作成します

`.creationDate : Date`

ファイルの作成日

`.creationTime : Time`

ファイルの作成時刻

`.delete()`

ファイルを削除します

`.exists : Boolean`

ディスク上にファイルが存在する場合は `true` を返します

`.extension : Text`

ファイル名の拡張子

`.fullName : Text`

拡張子 (あれば) を含めたファイルの完全な名称

`.getAppInfo() : Object`

`.exe` や `.dll`、`.plist` ファイルの情報をオブジェクトとして返します

`.getContent() : 4D.Blob`

ファイルの全コンテンツを格納した `4D.Blob` オブジェクトを返します

`.getIcon({ size : Integer }) : Picture`

ファイルのアイコンを返します

`.getText({ charSetName : Text { ; breakMode : Integer } }) : Text`

`getText($charSetName : Integer { ; $breakMode : Integer }) : Text`

.getRealCharSetNum : Integer { ; breakMode : Integer } } .text

ファイルのコンテンツをテキストとして返します

.hidden : Boolean

ファイルがシステムレベルで "非表示" に設定されていれば true

.isAlias : Boolean

ファイルがエイリアス、ショートカット、シンボリックリンクのいずれかである場合には true

.isFile : Boolean

ファイルに対しては常に true

.isFolder : Boolean

ファイルに対しては常に false

.isWritable : Boolean

ファイルがディスク上に存在し、書き込み可能な場合に true

.modificationDate : Date

ファイルの最終変更日

.modificationTime : Time

ファイルの最終変更時刻

.moveTo(destinationFolder : 4D.Folder { ; newName : Text }) : 4D.File

File オブジェクトを destinationFolder が指定する移行先へと移動すると同時に、newName を指定した場合は名称も変更します

.name : Text

拡張子 (あれば) を含まないファイル名

.original : 4D.File

.original : 4D.Folder

エイリアス、ショートカット、シンボリックリンクファイルのターゲット要素

.parent : 4D.Folder

対象ファイルの親フォルダーオブジェクト

.path : Text

ファイルの POSIX パス

.platformPath : Text

カレントプラットフォームのシンタクスで表現されたファイルのパス

.rename(newName : Text) : 4D.File

ファイル名を newName に指定した名称に変更し、名称変更後の File オブジェクトを返します

.setAppInfo(info : Object)

info に渡したプロパティを .exe や .dll、.plist ファイルの情報として書き込みます

.setContent (content : Blob)

content 引数の BLOB に保存されているデータを使用して、ファイルの全コンテンツを上書きします

.setText (text : Text {; charSetName : Text { ; breakMode : Integer } })

.setText (text : Text {; charSetNum : Integer { ; breakMode : Integer } })

text に渡されたテキストをファイルの新しいコンテンツとして書き込みます

.size : Real

ファイルのサイズ (バイト単位)

File

▶ 覆歴

File (*path* : Text { ; *pathType* : Integer }{ ; * }) : 4D.File

File (*fileConstant* : Integer { ; * }) : 4D.File

引数	タイプ	説明
<i>path</i>	Text	-> ファイルパス
<i>fileConstant</i>	Integer	-> 4Dファイル定数
<i>pathType</i>	Integer	-> <code>fk posix path</code> (デフォルト) または <code>fk platform path</code>
*		-> ホストデータベースのファイルを返すには * を渡します
戻り値	4D.File	<- 新規ファイルオブジェクト

説明

`File` コマンドは、`4D.File` 型の新しいオブジェクトを作成して返します。このコマンドは 2種類のシンタックスを受け入れます。

File (*path* { ; *pathType* } { ; * })

path には、ファイルパス文字列を渡します。カスタムの文字列やファイルシステム (例: "/DATA/myfile.txt") を渡すことができます。

`File` コマンドでは絶対パス名のみがサポートされます。

デフォルトで、4D は POSIXシンタックスで表現されたパスを期待します。プラットフォームパス名 (Windows または macOS) を使用する場合、*pathType* 引数を使用してそのことを宣言する必要があります。以下の定数を使用することができます:

定数	値	説明
<code>fk platform path</code>	1	プラットフォーム特有のシンタックスで表現されたパス (プラットフォームパス名の場合には必須)
<code>fk posix path</code>	0	POSIXシンタックスで表現されたパス (デフォルト)

File (*fileConstant* { ; * })

fileConstant には、以下の定数のどれか一つを指定して 4Dビルトインの、またはシステムファイルを渡します:

定数	値	説明
<code>Backup history file</code>	19	バックアップ履歴ファイル。バックアップ保存先フォルダに保存されています。
<code>Backup log file</code>	13	カレントのバックアップのログファイル。アプリケーションの Logs フォルダーに保存されています。
<code>Backup settings file</code>	1	プロジェクトの Settings フォルダーにある、デフォルトの backup.4DSettings ファイル (xml 形式)
<code>Backup settings file for data</code>	17	データフォルダーの Settings フォルダーにある、データファイル用の backup.4DSettings ファイル (xml 形式)
<code>Build application log file</code>	14	アプリケーションビルダーのカレントログファイル (xml 形式)。Logs フォルダーに保存されています。
<code>Build application settings file</code>	20	アプリケーションビルダーのデフォルト設定ファイル ("buildApp.4DSettings")。プロジェクトの Settings フォルダーに保存されています。
<code>Compacting</code>	6	Compact data file コマンドによって、あるいはメンテナンス&セキュリティセンター (MSC) によって作成された、直近の

log file 定数	値	説明
Current backup settings file	18	アプリケーションが現在使用している backup.4DSettings ファイル。使用されるのはデフォルトのバックアップ設定ファイル、または、データファイル用のユーザーバックアップ設定ファイルです。
Debug log file	12	SET DATABASE PARAMETER(Debug log recording) コマンドによって作成されたログファイル。Logs フォルダーに保存されています。
Diagnostic log file	11	SET DATABASE PARAMETER(Diagnostic log recording) コマンドによって作成されたログファイル。Logs フォルダーに保存されています。
Directory file	16	プロジェクトアプリケーションにおいて、ユーザーとグループ (あれば) の定義が格納された directory.json ファイル。このファイルは、データベースの user settings フォルダー (デフォルト、プロジェクトに対してグローバル)、または data settings フォルダー (データファイル専用) に保管されます。
HTTP debug log file	9	WEB SET OPTION(Web debug log) コマンドによって作成されたログファイル。Logs フォルダーに保存されています。
HTTP log file	8	WEB SET OPTION(Web log recording) コマンドによって作成されたログファイル。Logs フォルダーに保存されています。
IMAP Log file	23	SET DATABASE PARAMETER(IMAP Log) コマンドによって作成されたログファイル。Logs フォルダーに保存されています。
Last backup file	2	任意の場所に格納されている、最終バックアップファイル (名称は: <applicationName>[bkpNum].4BK)
Last journal integration log file	22	最後のログ統合ログファイル (あれば) の完全なパス名 (復元されたアプリケーションの Logs フォルダー内に保存されます)。このファイルは、自動修復モードにおいてログファイル統合が発生した時点で作成されます。
Repair log file	7	メンテナンス&セキュリティセンター (MSC) 内からデータベースに対しておこなわれたデータベース修復のログファイル。Logs フォルダーに保存されています。
Request log file	10	SET DATABASE PARAMETER(4D Server log recording) あるいは SET DATABASE PARAMETER(Client log recording) コマンドによって作成された標準のクライアント/サーバーログファイル (Web リクエストは除外)。サーバー上で実行された場合には、サーバーログが返されます (ログファイルはサーバー上の Logs フォルダーに保存されています)。クライアントで実行された場合には、クライアントのログが返されます (ログファイルはクライアントのLogsフォルダーに保存されています)。
SMTP log file	15	SET DATABASE PARAMETER(SMTP Log) コマンドによって作成されたログファイル。Logs フォルダーに保存されています。
User settings file	3	設定が有効化されている場合、ストラクチャーファイルと同じ階層にある Preferences フォルダーに格納された、全データファイルの settings.4DSettings ファイル。
User settings file for data	4	データファイルと同じ階層にある Preferences フォルダーに格納された、カレントデータファイルの settings.4DSettings ファイル。
Verification log file	5	VERIFY CURRENT DATA FILE および VERIFY DATA FILE コマンドによって、あるいはメンテナンス&セキュリティセンター (MSC) によって作成されたログファイル。Logs フォルダーに保存されています。

fileConstant 引数で指定したファイルが存在しない場合、null オブジェクトが返されます。エラーは生成されません。

コマンドがコンポーネントから呼び出されている場合、* 引数を渡してホストデータベースのパスを取得するようにします。* 引数を省略すると、常に null オブジェクトが返されます。

4D.File.new()

▶履歴

```
4D.File.new ( path : Text { ; pathType : Integer }{ ; * } ) : 4D.File
4D.File.new ( fileConstant : Integer { ; * } ) : 4D.File
```

説明

`4D.File.new()` 関数は、`4D.File` 型の新しいオブジェクトを作成して返します。この関数の機能は、`File` コマンドと同一です。

`4D.File.new()` よりも、短い `File` コマンドの使用が推奨されます。

.copyTo()

▶ 補足

`.copyTo(destinationFolder : 4D.Folder { ; newName : Text } { ; overwrite : Integer }) : 4D.File`

引数	タイプ		説明
destinationFolder	4D.Folder	->	宛先フォルダー
newName	Text	->	コピー先フォルダーの名前
overwrite	Integer	->	既存要素を上書きするには <code>fk overwrite</code> を渡します
戻り値	4D.File	<-	コピーされたファイル

説明

`.copyTo()` 関数は、`File` オブジェクトを、`destinationFolder` 引数で指定したフォルダへとコピーします。

`destinationFolder` 引数が指定するフォルダーはディスク上に存在している必要があります、そうでない場合にはエラーが生成されます。

デフォルトで、ファイルは元の名前を維持したままコピーされます。コピーの際にフォルダ名を変更したい場合、新しい名前を `newName` に渡します。新しい名前は命名規則に則っている必要があります（例：`:`, `/`, 等の文字を含んでいないなど）。そうでない場合、エラーが返されます。

`destinationFolder` 引数が指定するフォルダー内に同じ名前のファイルが既に存在する場合、4D はデフォルトでエラーを生成します。`overwrite` に `fk overwrite` 定数を渡すことで、既存のフォルダーを無視して上書きすることができます：

定数	値	説明
<code>fk overwrite</code>	4	既存要素があれば、それを上書きします

戻り値

コピーされた `File` オブジェクト。

例題

ユーザーのドキュメントフォルダーにあるピクチャーファイルを、アプリケーションフォルダー内にコピーします。

```
var $source; $copy : Object
$source:=Folder(fk documents folder).file("Pictures/photo.png")
$copy:=$source.copyTo(Folder("/PACKAGE");fk overwrite)
```

.create()

▶ 補足

ZIPアーカイブには利用できません

`.create() : Boolean`

引数	タイプ		説明
戻り値	Boolean	<-	ファイルが正常に作成された場合に <code>true</code> 、それ以外の場合は <code>false</code>

説明

`.create()` 関数は、`File` オブジェクトのプロパティに基づいてディスク上にファイルを作成します。

必要であれば、関数は `platformPath` あるいは `path` プロパティの詳細に基づいてフォルダー階層を作成します。ファイルがディスク上にすでに存在する場合、関数は何もせず、`false` を返します（エラーは返されません）。

戻り値

- ファイルが正常に作成された場合には `true`
- すでに同じ名前のファイルが存在する、あるいはエラーが発生した場合には `false`

例題

データベースフォルダー内にプリファレンスファイルを作成します：

```
var $created : Boolean  
$created:=File("/PACKAGE/SpecialPrefs/"+Current user+".myPrefs").create()
```

.createAlias()

▶ 補足

`.createAlias(destinationFolder : 4D.Folder ; aliasName : Text { ; aliasType : Integer }) : 4D.File`

引数	タイプ		説明
destinationFolder	4D.Folder	->	エイリアスまたはショートカットの作成先フォルダー
aliasName	Text	->	エイリアスまたはショートカットの名称
aliasType	Integer	->	エイリアスリンクのタイプ
戻り値	4D.File	<-	エイリアスまたはショートカットのファイル参照

説明

`.createAlias()` 関数は、`destinationFolder` オブジェクトで指定されたフォルダー内に、`aliasName` が指定する名称で、対象ファイルへのエイリアス（macOS）またはショートカット（Windows）を作成します。

`aliasName` には、作成するエイリアスまたはショートカットの名前を渡します。

macOS 上では、この関数はデフォルトで標準エイリアスを作成します。`aliasType` 引数を渡すことで、シンボリックリンクを作成することもできます。以下の定数を使用することができます：

定数	値	説明
<code>fk alias link</code>	0	エイリアスリンク（デフォルト）
<code>fk symbolic link</code>	1	シンボリックリンク（macOSのみ）

Windows 上では、常にショートカット (.lnk ファイル) が作成されます (`aliasType` 引数は無視されます)。

返されるオブジェクト

`isAlias` プロパティが `true` に設定された `4D.File` オブジェクトを返します。

例題

データベースフォルダー内のファイルへのエイリアスを作成します：

```
$myFile:=Folder(fk documents folder).file("Archives/ReadMe.txt")  
$aliasFile:=$myFile.createAlias(File("/PACKAGE");"ReadMe")
```

.creationDate

▶ 補足

.creationDate : Date

説明

.creationDate プロパティは、ファイルの作成日を返します。

このプロパティは 読み取り専用 です。

.creationTime

▶ 履歴

.creationTime : Time

説明

.creationTime プロパティは、ファイルの作成時刻 を返します (00:00からの経過秒数の形式)。

このプロパティは 読み取り専用 です。

.delete()

▶ 履歴

.delete()

| 引数 | タイプ | | 説明 | | -- | --- | | ----- | | | | このコマンドは引数を必要としません |

説明

.delete() 関数は、ファイルを削除します。

ファイルが現在開いている場合、エラーが生成されます。

ファイルがディスク上に存在しない場合、関数は何もしません (エラーは何も生成されません)。

警告: .delete() はディスク上の任意のファイルを削除することができます。これには、他のアプリケーションで作成されたドキュメントや、アプリケーションそのものも対象になります。そのため、.delete() は特に十分な注意を払って使用してください。ファイルの削除は恒久的な操作であり取り消しできません。

例題

データベースフォルダー内の特定のファイルを削除します:

```
$tempo:=File("/PACKAGE/SpecialPrefs/"+Current user+".prefs")
If($tempo.exists)
    $tempo.delete()
    ALERT("ユーザーのプリファレンスファイルが削除されました。")
End if
```

.exists

▶ 履歴

.exists : Boolean

説明

.exists プロパティは、ディスク上にファイルが存在する場合は true を返します(それ以外の場合は false)。

このプロパティは 読み取り専用 です。

.extension

▶ 履歴

.extension : Text

説明

.extension プロパティは、ファイル名の拡張子を返します（あれば）。拡張子は必ず"."で始まります。ファイル名が拡張子を持たない場合には、このプロパティは空の文字列を返します。

このプロパティは 読み取り専用 です。

.fullName

▶ 履歴

.fullName : Text

説明

.fullName プロパティは、拡張子（あれば）を含めたファイルの完全な名称を返します。

このプロパティは 読み取り専用 です。

.getAppInfo()

▶ 履歴

.getAppInfo() : Object

引数	タイプ	説明
戻り値	Object	<- .exe/.dll のバージョンリソースや.plist ファイルの中身

説明

.getAppInfo() 関数は、.exe や .dll、.plist ファイルの情報をオブジェクトとして返します。

この関数は、既存の .exe、.dll、あるいは .plist ファイルと使う必要があります。ファイルがディスク上に存在しない、または、有効な .exe や .dll、.plist ファイルでない場合、この関数は空のオブジェクトを返します（エラーは生成されません）。

この関数は xml形式の.plist ファイル（テキスト）のみをサポートしています。バイナリ形式の.plist ファイルを対象に使用した場合、エラーが返されます。

.exe または .dll ファイルの場合に返されるオブジェクト

.exe および .dll ファイルの読み取りは Windows 上でのみ可能です。

プロパティはすべてテキストです。

プロパティ	タイプ
InternalName	Text
ProductName	Text
CompanyName	Text
LegalCopyright	Text
ProductVersion	Text
FileDescription	Text
FileVersion	Text
OriginalFilename	Text

.plist ファイルの場合に返されるオブジェクト

xml ファイルの中身は解析され、オブジェクトのプロパティとしてキーが返されます。キーの型（テキスト、布尔、数値）は維持されます。 .plist dict は JSON オブジェクトとして返されます。また、 .plist array は JSON 配列として返されます。

例題

```
// アプリケーションの .exe ファイルの著作権情報を表示します (Windows)
var $exeFile : 4D.File
var $info : Object
$exeFile:=File(Application file; fk platform path)
$info:=$exeFile.getAppInfo()
ALERT($info.LegalCopyright)

// info.plist の著作権情報を表示します (Windows および macOS)
var $infoPlistFile : 4D.File
var $info : Object
$infoPlistFile:=File("/RESOURCES/info.plist")
$info:=$infoPlistFile.getAppInfo()
ALERT($info.Copyright)
```

参照

[.setAppInfo\(\)](#)

.getContent()

▶履歴

.getContent() : 4D.Blob

引数	タイプ		説明
戻り値	4D.Blob	<-	ファイルのコンテンツ

説明

.getContent() 関数は、ファイルの全コンテンツを格納した 4D.Blob オブジェクトを返します。BLOB についての詳細は、BLOB の章を参照してください。

戻り値

4D.Blob オブジェクト。

例題

ドキュメントの中身を BLOB フィールドに保存します：

```

var $vPath : Text
$vPath:=Select document("");*"Select a document";0)
If(OK=1) // キュメントが選択されていれば
[aTable]aBlobField:=File($vPath;fk platform path).getContent()
End if

```

.getIcon()

▶ 覆歴

.getIcon({ size : Integer }) : Picture

引数	タイプ		説明
size	Integer	->	取得するピクチャーの一辺の長さ (ピクセル単位)
戻り値	Picture	<-	アイコン

説明

.getIcon() 関数は、ファイルのアイコンを返します。

任意の size 引数を渡すと、返されるアイコンのサイズをピクセル単位で指定することができます。この値は、実際にはアイコンを格納している正方形の一辺の長さを表しています。アイコンは通常、32x32ピクセル ("大きいアイコン") または 16x16ピクセル ("小さいアイコン") で定義されています。この引数に 0 を渡すか省略した場合、"大きいアイコン" が返されます。

ファイルがディスク上に存在しない場合、デフォルトの空のアイコンが返されます。

戻り値

ファイルアイコンの [ピクチャー](#)。

.getText()

▶ 覆歴

.getText({ charSetName : Text { ; breakMode : Integer } }) : Text
.getText({ charSetNum : Integer { ; breakMode : Integer } }) : Text

引数	タイプ		説明
charSetName	Text	->	文字セットの名前
charSetNum	Integer	->	文字セットの番号
breakMode	Integer	->	改行の処理モード
戻り値	Text	<-	ドキュメントから取得したテキスト

説明

.getText() 関数は、ファイルのコンテンツをテキストとして返します。

任意で、コンテンツの読み取りに使用する文字セットを渡します。これには、次の二つの方法があります:

- `charSetName` に標準の文字セット名を含んだ文字列 ("ISO-8859-1" や "UTF-8" など) を渡します。
- `charSetNum` に標準の文字セット名の MIBEnum ID (倍長整数) を渡します。

4D によってサポートされている文字セットの一覧については、 [CONVERT FROM TEXT](#) コマンドを参照ください。

ドキュメントにバイトオーダーマーク (BOM) が含まれている場合、4D は `charSetName` または `charSetNum` 引数で設定されている文字セットではなく、BOM で指定されたものを使用します (結果として引数は無視されます)。ドキュメントに BOM が含まれておらず、また `charSetName` および `charSetNum` 引数が渡されなかった場合、4D はデフォルトで "UTF-8" を文字セットとして使用します。

breakMode には、ドキュメントの改行文字に対しておこなう処理を指定する倍長整数を渡します。"System Documents" テーマの、以下の定数を使用することができます:

定数	値	説明
Document unchanged	0	何も処理をしません。
Document with native format	1	(デフォルト) 改行は OS のネイティブフォーマットに変換されます。macOS では CR (キャリッジリターン) に、Windows では CRLF (キャリッジリターン+ラインフィード) に変換されます。
Document with CRLF	2	改行は Windowsフォーマット (CRLF、キャリッジリターン+ラインフィード) へと変換されます。
Document with CR	3	改行は macOSフォーマット (CR、キャリッジリターン) へと変換されます。
Document with LF	4	改行は Unixフォーマット (LF、ラインフィード) へと変換されます。

breakMode 引数を渡さなかった場合はデフォルトで、改行はネイティブモード (1) で処理されます。

戻り値

ファイルのテキスト。

例題

以下のテキストを持つドキュメントがある場合を考えます (フィールドはタブ区切りです):

```
id name price vat
3 thé 1.06€ 19.6
2 café 1.05€ 19.6
```

以下のコードを実行すると:

```
$myFile:=Folder(fk documents folder).file("Billing.txt") // デフォルトでUTF-8
$txt:=$myFile.getText()
```

以下の結果が `$txt` に得られます:

```
"id\tname\tprice\tvat\r\n3\tthé\t1.06€\t19.6\r\n2\tcafé\t1.05€\t19.6"
```

このとき、区切り文字は `\t` (タブ) で、改行コードは `\r\n` (CRLF) です。

以下は、同じファイルで改行コードが異なる例です:

```
$txt:=$myFile.getText("UTF-8", Document with LF)
```

この場合、`$txt` の値は次の通りです:

```
"id\tname\tprice\tvat\r\n3\tthé\t1.06€\t19.6\t\tcafé\t1.05€\t19.6"
```

このとき、改行コードは `\n` (LF) です。

.hidden

▶ 履歴

.hidden : Boolean

説明

`.hidden` プロパティは、ファイルがシステムレベルで "非表示" に設定されていれば `true`を返します (それ以外の場合は `false`)。

このプロパティは 読み取り専用 です。

.isAlias

▶ 履歴

`.isAlias` : Boolean

説明

`.isAlias` プロパティは、ファイルがエイリアス、ショートカット、シンボリックリンクのいずれかである場合には `true`を返し、それ以外の場合には `false` を返します。

このプロパティは 読み取り専用 です。

.isFile

▶ 履歴

`.isFile` : Boolean

説明

`.isFile` プロパティは、ファイルに対しては常に `true`を返します。

このプロパティは 読み取り専用 です。

.isFolder

▶ 履歴

`.isFolder` : Boolean

説明

`.isFolder` プロパティは、ファイルに対しては常に `false`を返します。

このプロパティは 読み取り専用 です。

.isWritable

▶ 履歴

`.isWritable` : Boolean

説明

`.isWritable` プロパティは、ファイルがディスク上に存在し、書き込み可能な場合に `true`を返します。

このプロパティは 4Dアプリケーションがディスクに書き入めるかどうか (アクセス権限) をチェックし、ファイルの `writable` (書き込み可能) 属性のみ依存するわけではありません。

このプロパティは 読み取り専用 です。

例題

```
$myFile:=File("C:\\Documents\\Archives\\ReadMe.txt";fk platform path)
If($myFile.isWritable)
    $myNewFile:=$myFile.setText("Added text")
End if
```

.modificationDate

▶ 覆歴

.modificationDate : Date

説明

.modificationDate プロパティは、ファイルの最終変更日を返します。

このプロパティは 読み取り専用 です。

.modificationTime

▶ 覆歴

.modificationTime : Time

説明

.modificationTime プロパティは、ファイルの最終変更時刻 を返します (00:00 からの経過秒数の形式)。

このプロパティは 読み取り専用 です。

.moveTo()

▶ 覆歴

.moveTo(destinationFolder : 4D.Folder { ; newName : Text }) : 4D.File

引数	タイプ		説明
destinationFolder	4D.Folder	->	宛先フォルダー
newName	Text	->	移動先でのファイルの完全な名称
戻り値	4D.File	<-	移動したファイル

説明

.moveTo() 関数は、 File オブジェクトを destinationFolder が指定する移行先へと移動すると同時に、newName を指定した場合は名称も変更します。

destinationFolder 引数が指定するフォルダーはディスク上に存在している必要があります、そうでない場合にはエラーが生成されます。

デフォルトで、移動したファイルは元の名前を維持します。移動の際にファイル名を変更したい場合、新しい完全な名前を newName に渡します。新しい名前は命名規則に則っている必要があります (例: ":"、"/"、等の文字を含んでいない、など)。そうでない場合、エラーが返されます。

返されるオブジェクト

移動後の File オブジェクト。

例題

```
$DocFolder:=Folder(fk documents folder)
$myFile:=$DocFolder.file("Current/Infos.txt")
$myFile.moveTo($DocFolder.folder("Archives");"Infos_old.txt")
```

.name

▶ 覆歴

.name : Text

説明

.name プロパティは、拡張子（あれば）を含まないファイル名を返します。

このプロパティは 読み取り専用 です。

.original

▶ 履歴

.original : 4D.File

.original : 4D.Folder

説明

.original プロパティは、エイリアス、ショートカット、シンボリックリンクファイルのターゲット要素を返します。ターゲット要素は以下のいずれかです：

- File オブジェクト
- Folder オブジェクト

エイリアスでないファイルについては、プロパティは同じファイルオブジェクトをファイルとして返します。

このプロパティは 読み取り専用 です。

.parent

▶ 履歴

.parent : 4D.Folder

説明

.parent プロパティは、対象ファイルの親フォルダーオブジェクトを返します。パスがシステムパスを表す場合（例："/DATA/"）、システムパスが返されます。

このプロパティは 読み取り専用 です。

.path

▶ 履歴

.path : Text

説明

.path プロパティは、ファイルの POSIX パスを返します。パスがファイルシステムを表す場合（例："/DATA/"）、ファイルシステムが返されます。

このプロパティは 読み取り専用 です。

.platformPath

▶ 履歴

.platformPath : Text

説明

.platformPath プロパティは、カレントプラットフォームのシンタックスで表現されたファイルのパスを返します。

このプロパティは 読み取り専用 です。

.rename()

▶ 履歴

.rename(newName : Text) : 4D.File

引数	タイプ		説明
newName	Text	->	ファイルの新しい完全な名称
戻り値	4D.File	<-	名称変更されたファイル

説明

.rename() 関数は、ファイル名を *newName* に指定した名称に変更し、名称変更後の File オブジェクトを返します。

newName 引数は命名規則に則っている必要があります（例："："、"/"、等の文字を含んでいない、など）。そうでない場合、エラーが返されます。同じ名前のファイルがすでに存在する場合には、エラーが返されます。

この関数はファイルの完全な名前を編集することに留意が必要です。つまり、*newName* に拡張子を渡さなかった場合、新しいファイル名には拡張子がありません。

返されるオブジェクト

名称変更された File オブジェクト。

例題

"ReadMe.txt" ファイルを "ReadMe_new.txt" というファイルに名称変更します：

```
$toRename:=File("C:\\\\Documents\\\\Archives\\\\ReadMe.txt";fk platform path)
$newName:=$toRename.rename($toRename.name+"_new"+$toRename.extension)
```

.setAppInfo()

▶履歴

.setAppInfo(*info* : Object)

引数	タイプ		説明
<i>info</i>	Object	->	.exe/.dll のバージョンリソースや .plist ファイルに書き込むプロパティ

説明

.setAppInfo() 関数は、*info* に渡したプロパティを .exe や .dll、.plist ファイルの情報として書き込みます。

この関数は、既存の .exe、.dll、あるいは .plist ファイルと使う必要があります。ファイルがディスク上に存在しない、または、有効な .exe や .dll、.plist ファイルでない場合、この関数は何もしません（エラーは生成されません）。

この関数は xml 形式の .plist ファイル（テキスト）のみをサポートしています。バイナリ形式の .plist ファイルを対象に使用した場合、エラーが返されます。

.exe または .dll ファイル用の *info* オブジェクト

.exe および .dll ファイル情報の書き込みは Windows 上でのみ可能です。

info オブジェクトに設定された各プロパティは .exe または .dll ファイルのバージョンリソースに書き込まれます。以下のプロパティが使用できます（それ以外のプロパティは無視されます）：

プロパティ	タイプ
InternalName	Text
ProductName	Text
CompanyName	Text
LegalCopyright	Text
ProductVersion	Text
FileDescription	Text
FileVersion	Text
OriginalFilename	Text

値として null または空テキストを渡すと、空の文字列がプロパティに書き込まれます。テキストでない型の値を渡した場合には、文字列に変換されます。

.plist ファイル用の *info* オブジェクト

info オブジェクトに設定された各プロパティは .plist ファイルにキーとして書き込まれます。あらゆるキーの名称が受け入れられます。値の型は可能な限り維持されます。

info に設定されたキーが .plist ファイル内すでに定義されている場合は、その値が更新され、元の型が維持されます。.plist ファイルに既存のそのほかのキーはそのまま維持されます。

日付型の値を定義するには、Xcode plist エディターのようにミリ秒を除いた ISO UTC 形式の JSON タイムスタンプ文字列（例："2003-02-01T01:02:03Z"）を使用します。

例題

```
// .exe ファイルの著作権およびバージョン情報を設定します (Windows)
var $exeFile : 4D.File
var $info : Object
$exeFile:=File(Application file; fk platform path)
$info:=New object
$info.LegalCopyright:="Copyright 4D 2021"
$info.ProductVersion:="1.0.0"
$exeFile.setAppInfo($info)
```

```
// info.plist ファイルのキーをいくつか設定します (Windows および macOS)
var $infoPlistFile : 4D.File
var $info : Object
$infoPlistFile:=File("/RESOURCES/info.plist")
$info:=New object
$info.Copyright:="Copyright 4D 2021" // テキスト
$info.ProductVersion:=12 // 整数
$info.ShipmentDate:="2021-04-22T06:00:00Z" // タイムスタンプ
$infoPlistFile.setAppInfo($info)
```

参照

[.getAppInfo\(\)](#)

[.setContent\(\)](#)

▶ 覆歴

[.setContent \(content : Blob \)](#)

引数	タイプ		説明
content	BLOB	->	ファイルの新しいコンテンツ

説明

`.setContent()` 関数は、`content` 引数の BLOB に保存されているデータを使用して、ファイルの全コンテンツを上書きします。BLOB についての詳細は、[BLOB](#) の章を参照してください。

例題

```
$myFile:=Folder(fk documents folder).file("Archives/data.txt")
$myFile.setContent([aTable]aBlobField)
```

.setText()

▶ 履歴

`.setText(text : Text {; charSetName : Text { ; breakMode : Integer } })`
`.setText(text : Text {; charSetNum : Integer { ; breakMode : Integer } })`

引数	タイプ		説明
テキスト	Text	->	ファイルに保存するテキスト
charSetName	Text	->	文字セットの名前
charSetNum	Integer	->	文字セットの番号
breakMode	Integer	->	改行の処理モード

説明

`.setText()` 関数は、`text` に渡されたテキストをファイルの新しいコンテンツとして書き込みます。

`File` オブジェクトで参照されているファイルがディスク上に存在しない場合、このメソッドがそのファイルを作成します。ディスク上にファイルが存在する場合、ファイルが開かれている場合を除き、以前のコンテンツは消去されます。ファイルが開かれている場合はコンテンツはロックされ、エラーが生成されます。

`text` には、ファイルに書き込むテキストを渡します。テキストリテラル ("my text" など) のほか、4Dテキストフィールドや変数も渡せます。

任意で、コンテンツの書き込みに使用する文字セットを渡します。これには、次の二つの方法があります：

- `charSetName` に標準の文字セット名を含んだ文字列 ("ISO-8859-1" や "UTF-8" など) を渡します。
- `charSetNum` に標準の文字セット名の MIBEnum ID (倍長整数) を渡します。

4D によってサポートされている文字セットの一覧については、[CONVERT FROM TEXT](#) コマンドを参照ください。

文字セットにバイトオーダーマーク (BOM) が存在し、かつその文字セットに "-no-bom" 接尾辞 (例: "UTF-8-no-bom") が含まれていない場合、4D は BOM をファイルに挿入します。文字セットを指定しない場合、4D はデフォルトで "UTF-8" の文字セットを BOMなしで使用します。

`breakMode` には、ファイルを保存する前に改行文字に対しておこなう処理を指定する倍長整数を渡します。System Documents テーマ内にある、以下の定数を使用することができます：

定数	値	説明
Document unchanged	0	何も処理をしません。
Document with native format	1	(デフォルト) 改行は OS のネイティブフォーマットに変換されます。macOS では LF (ラインフィード) に、Windows では CRLF (キャリッジリターン+ラインフィード) に変換されます。
Document with CRLF	2	改行は Windows のデフォルトフォーマットである CRLF (キャリッジリターン+ラインフィード) へと変換されます。
Document with CR	3	改行はクラシック Mac OS のデフォルトフォーマットである CR (キャリッジリターン) へと変換されます。
Document with LF	4	改行は Unix および macOS のデフォルトフォーマットである LF (ラインフィード) へと変換されます。

breakMode 引数を渡さなかった場合はデフォルトで、改行はネイティブモード (1) で処理されます。

互換性に関する注記: EOL (改行コード) および BOM の管理については、互換性オプションが利用可能です。doc.4d.com の [互換性ページ](#) を参照ください。

例題

```
$myFile:=File("C:\\Documents\\Hello.txt";fk platform path)
$myFile.setText("Hello world")
```

.size

▶ 履歴

.size : Real

説明

.size プロパティは、ファイルのサイズ (バイト単位)を返します。ファイルがディスク上に存在しない場合、サイズは 0 になります。

このプロパティは 読み取り専用 です。

Folder

`Folder` オブジェクトは `Folder` コマンドによって作成されます。これらのオブジェクトには、(実在しているか否かに関わらず) フォルダーへの参照が格納されます。たとえば、新規フォルダーを作成するために `Folder` コマンドを実行した場合、有効な `Folder` オブジェクトが作成されますが、`folder.create()` 関数を呼び出すまで、ディスク上にはなにも保存されていません。

例題

"JohnSmith" フォルダーを作成します:

```
Form.curfolder:=Folder(fk database folder)
Form.curfolder:=Folder("C:\\Users\\JohnSmith\\";fk platform path)
```

Folder オブジェクト

`.copyTo(destinationFolder : 4D.Folder { ; newName : Text } { ; overwrite : Integer }) : 4D Folder`

`Folder` オブジェクトを、`destinationFolder` 引数で指定したフォルダーへとコピーします

`.create() : Boolean`

`Folder` オブジェクトのプロパティに基づいてディスク上にフォルダーを作成します

`.createAlias(destinationFolder : 4D.Folder ; aliasName : Text { ; aliasType : Integer }) : 4D.File`

エイリアス (macOS) またはショートカット (Windows) を作成します

`.creationDate : Date`

フォルダーの作成日

`.creationTime : Time`

フォルダーの作成時刻

`.delete({ option : Integer })`

フォルダーを削除します

`.exists : Boolean`

ディスク上にフォルダーが存在する場合は `true`

`.extension : Text`

フォルダー名の拡張子

`.file(path : Text) : 4D.File`

`Folder` オブジェクト内に `File` オブジェクトを作成し、その参照を返します

`.files({ options : Integer }) : Collection`

フォルダーに格納されている `File` オブジェクトのコレクションを返します

`.folder(path : Text) : 4D.Folder`

親の `Folder` オブジェクト内に新しい `Folder` オブジェクトを作成し、その参照を返します

`.folders({ options : Integer }) : Collection`

親フォルダーに格納されている `Folder` オブジェクトのコレクションを返します

`.fullName : Text`

フルネーム (ドライブ名 + パス + ファイル名)

.getIcon({ size : Integer }) : Picture	拡張子 (あれば) を含めにフォルダーの元的な名前
.hidden : Boolean	フォルダーのアイコンを返します
.isAlias : Boolean	フォルダーがシステムレベルで "非表示" に設定されていれば true
.isFile : Boolean	Folder オブジェクトに対しては常に false
.isFolder : Boolean	フォルダーに対しては常に true
.isPackage : Boolean	フォルダーが macOS上のパッケージである (かつディスク上に存在している) 場合に true
.modificationDate : Date	フォルダーの最終変更日
.modificationTime : Time	フォルダーの最終変更時刻
.name : Text	拡張子 (あれば) を含まないフォルダーナン
.original : 4D.Folder	対象フォルダーと同じフォルダーオブジェクト
.parent : 4D.Folder	対象フォルダーの親フォルダーオブジェクト
.path : Text	フォルダーの POSIXパス
.platformPath : Text	カレントプラットフォームのシンタックスで表現されたフォルダーのパス
.moveTo(destinationFolder : 4D.Folder { ; newName : Text }) : 4D.Folder	Folder オブジェクト (ソースフォルダー) を destinationFolder が指定する移行先へと移動すると同時に、newName を指定した場合は名称も変更します
.rename(newName : Text) : 4D.Folder	フォルダーナンを newName に指定した名称に変更し、名称変更後の Folder オブジェクトを返します

Folder

▶履歴

Folder (path : Text { ; pathType : Integer }{ ; * }) : 4D.Folder

Folder (folderConstant : Integer { ; * }) : 4D.Folder

引数	タイプ		説明
path	Text	->	フォルダーパス
folderConstant	Integer	->	4Dフォルダー定数
pathType	Integer	->	<code>fk posix path</code> (デフォルト) または <code>fk platform path</code>
*		->	ホストデータベースのフォルダーを返すには * を渡します
戻り値	4D.Folder	<-	新規フォルダーオブジェクト

説明

`Folder` コマンドは、`4D.Folder` 型の新しいオブジェクトを作成して返します。このコマンドは 2種類のシンタックスを受け入れます。

`Folder (path { ; pathType } { ; * })`

path には、フォルダーパス文字列を渡します。カスタムの文字列やファイルシステム (例: "/DATA") を渡すことができます。

`Folder` コマンドでは絶対パス名のみがサポートされます。

デフォルトで、4D は POSIXシンタックスで表現されたパスを期待します。プラットフォームパス名 (Windows または macOS) を使用する場合、*pathType* 引数を使用してそのことを宣言する必要があります。以下の定数を使用することができます:

定数	値	説明
<code>fk platform path</code>	1	プラットフォーム特有のシンタックスで表現されたパス (プラットフォームパス名の場合には必須)
<code>fk posix path</code>	0	POSIXシンタックスで表現されたパス (デフォルト)

`Folder (folderConstant { ; * })`

folderConstant には、以下の定数のどれか一つを指定して 4Dビルトインの、またはシステムフォルダーを渡します:

定数	値	説明
<code>fk applications folder</code>	116	
<code>fk data folder</code>	9	関連づけられたファイルシステム: "/DATA"
<code>fk database folder</code>	4	関連づけられたファイルシステム: "/PACKAGE"
<code>fk desktop folder</code>	115	
<code>fk documents folder</code>	117	ユーザーのドキュメントフォルダー
<code>fk licenses folder</code>	1	マシンの 4Dライセンスファイルを格納しているフォルダー
<code>fk logs folder</code>	7	関連づけられたファイルシステム: "/LOGS"
<code>fk mobileApps folder</code>	10	
<code>fk remote database folder</code>	3	それぞれの 4Dリモートマシン上に作成された 4Dデータベースフォルダー
<code>fk resources folder</code>	6	関連づけられたファイルシステム: "/RESOURCES"
<code>fk system folder</code>	100	
<code>fk user preferences folder</code>	0	ユーザー環境設定ファイルを <userName> ディレクトリに保存している 4Dフォルダー
<code>fk web root folder</code>	8	データベースのカレントの Webルートフォルダー: ただし "/PACKAGE/path" のパッケージ内にある場合。 そうでない場合はフルパス。

コマンドがコンポーネントから呼び出されている場合、* 引数を渡してホストデータベースのパスを取得するようにします。* 引数を省略すると、常に null オブジェクトが返されます。

4D.Folder.new()

▶履歴

4D.Folder.new (*path* : Text { ; *pathType* : Integer }{ ; * }) : 4D.Folder

4D.Folder.new (*folderConstant* : Integer { ; * }) : 4D.Folder

説明

`4D.Folder.new()` 関数は、`4D.Folder` 型の新しいオブジェクトを作成して返します。この関数の機能は、`Folder` コマンドと同一です。

`4D.Folder.new()` よりも、短い `Folder` コマンドの使用が推奨されます。

.copyTo()

▶履歴

`.copyTo(destinationFolder : 4D.Folder { ; newName : Text } { ; overwrite : Integer }) : 4D.Folder`

引数	タイプ		説明
<i>destinationFolder</i>	4D.Folder	->	宛先フォルダー
<i>newName</i>	Text	->	コピー先フォルダーの名前
<i>overwrite</i>	Integer	->	既存要素を上書きするには <code>fk overwrite</code> を渡します
戻り値	4D.Folder	<-	コピーされたフォルダー

説明

`.copyTo()` 関数は、`Folder` オブジェクトを、*destinationFolder* 引数で指定したフォルダーへとコピーします。

destinationFolder 引数が指定するフォルダーはディスク上に存在している必要があります、そうでない場合にはエラーが生成されます。

デフォルトで、フォルダーは元の名前を維持したままコピーされます。コピーの際にフォルダーネームを変更したい場合、新しい名前を *newName* に渡します。新しい名前は命名規則に則っている必要があります（例： `:`、`/`、等の文字を含んでいない、など）。そうでない場合、エラーが返されます。

destinationFolder 引数が指定するフォルダー内に同じ名前のフォルダーが既に存在する場合、4D はデフォルトでエラーを生成します。*overwrite* に `fk overwrite` 定数を渡すことで、既存のフォルダーを無視して上書きすることができます：

定数	値	説明
<code>fk overwrite</code>	4	既存要素があれば、それを上書きします

戻り値

コピーされた `Folder` オブジェクト。

例題

ユーザーのドキュメントフォルダーにあるピクチャーフォルダーを、データベースフォルダー内にコピーします。

```
var $userImages; $copiedImages : 4D.Folder  
$userImages:=Folder(fk documents folder+"/Pictures/")  
$copiedImages:=$userImages.copyTo(Folder(fk database folder);fk overwrite)
```

.create()

▶履歴

`.create()` : Boolean

引数	タイプ		説明
戻り値	Boolean	<-	フォルダーが正常に作成された場合には true、それ以外の場合は false

説明

`.create()` 関数は、`Folder` オブジェクトのプロパティに基づいてディスク上にフォルダーを作成します。

必要であれば、関数は `platformPath` あるいは `path` プロパティの詳細に基づいてフォルダー階層を作成します。フォルダーがディスク上にすでに存在する場合、関数は何もせず、`false` を返します（エラーは返されません）。

戻り値

- フォルダーが正常に作成された場合には `true`
- すでに同じ名前のフォルダーが存在する、あるいはエラーが発生した場合には `false`

例題 1

データベースフォルダー内に空のフォルダーを作成します：

```
var $created : Boolean
$created:=Folder("/PACKAGE/SpecialPrefs").create()
```

例題 2

データベースフォルダー内に "/Archives2019/January/" フォルダーを作成します：

```
$newFolder:=Folder("/PACKAGE/Archives2019/January")
If($newFolder.create())
    ALERT($newFolder.name+" フォルダーが作成されました。")
Else
    ALERT($newFolder.name+" フォルダーは作成できませんでした。")
End if
```

.createAlias()

▶ 覆歴

`.createAlias(destinationFolder : 4D.Folder ; aliasName : Text { ; aliasType : Integer }) : 4D.File`

引数	タイプ		説明
destinationFolder	4D.Folder	->	エイリアスまたはショートカットの作成先フォルダー
aliasName	Text	->	エイリアスまたはショートカットの名称
aliasType	Integer	->	エイリアスリンクのタイプ
戻り値	4D.File	<-	エイリアスまたはショートカットのフォルダー参照

説明

`.createAlias()` 関数は、`destinationFolder` オブジェクトで指定されたフォルダー内に、`aliasName` が指定する名称で、対象フォルダーへのエイリアス（macOS）またはショートカット（Windows）を作成します。

`aliasName` には、作成するエイリアスまたはショートカットの名前を渡します。

macOS 上では、この関数はデフォルトで標準エイリアスを作成します。`aliasType` 引数を渡すことで、シンボリックリンクを作成することもできます。以下の定数を使用することができます：

定数	値	説明
fk alias link	0	エイリアスリンク (デフォルト)
fk symbolic link	1	シンボリックリンク (macOSのみ)

Windows 上では、常にショートカット (.lnk ファイル) が作成されます (*aliasType* 引数は無視されます)。

返されるオブジェクト

`isAlias` プロパティが `true` に設定された `4D.File` オブジェクトを返します。

例題

データベースフォルダー内のアーカイブフォルダーへのエイリアスを作成します:

```
$myFolder:=Folder("C:\\\\Documents\\\\Archives\\\\2019\\\\January";fk platform path)
$aliasFile:=$myFolder.createAlias(Folder("/PACKAGE");"Jan2019")
```

.creationDate

▶ 覆歴

`.creationDate` : Date

説明

`.creationDate` プロパティは、フォルダーの作成日を返します。

このプロパティは 読み取り専用 です。

.creationTime

▶ 覆歴

`.creationTime` : Time

説明

`.creationTime` プロパティは、フォルダーの作成時刻 を返します (00:00 からの経過秒数の形式)。

このプロパティは 読み取り専用 です。

.delete()

▶ 覆歴

`.delete({ option : Integer })`

引数	タイプ		説明
option	Integer	->	フォルダー削除のオプション

説明

`.delete()` 関数は、フォルダーを削除します。

セキュリティ上の理由から、option 引数を渡さなかった場合はデフォルトで、`.delete()` は空のフォルダーしか削除しません。 空でないフォルダーを削除するには、以下の定数のいずれか一つを option 引数として渡す必要があります:

定数	値	説明
Delete only if empty	0	フォルダーが空の場合のみ削除します
Delete with contents	1	フォルダーを中身ごと削除します

`Delete only if empty` が渡された、または option 引数を渡さなかった場合:

- フォルダーが空の場合にしか削除されません。そうでない場合、コマンドは何もせず、エラー-47 が生成されます。
- フォルダーが存在しない場合、エラー-120 が生成されます。

`Delete with contents` を渡した場合:

- フォルダーと、その中身がすべて削除されます。警告: フォルダーまたはその中身がロックされている、あるいは読み取り専用に設定されていたとしても、カレントユーザーが適切なアクセス権を持っていた場合には、フォルダーはその中身ごと削除されます。
- このフォルダー、またはその中のフォルダーのどいどれかが削除できなかった場合、削除できない要素が検知された時点で削除は中止され、エラー(*) が返されます。このとき、フォルダーは途中までしか削除されていない可能性があります。削除が中止された場合、`GET LAST ERROR STACK` コマンドを使用して原因となったファイルの名前とパスを取得することができます。
- フォルダーが存在しない場合、コマンドは何もせず、エラーは返されません。

(*) Windowsの場合: -54 (ロックされたファイルを書き込みのために開こうとした)

macOSの場合: -45 (ファイルはロックされていたか不正なパス名)

.exists

▶履歴

.exists : Boolean

説明

`.exists` プロパティは、ディスク上にフォルダーが存在する場合は `true`を返します (それ以外の場合は `false`)。

このプロパティは 読み取り専用 です。

.extension

▶履歴

.extension : Text

説明

`.extension` プロパティは、フォルダー名の拡張子を返します (あれば)。拡張子は必ず". "で始まります。フォルダー名が拡張子を持たない場合には、このプロパティは空の文字列を返します。

このプロパティは 読み取り専用 です。

.file()

▶履歴

.file(`path` : Text) : 4D.File

引数	タイプ		説明
<code>path</code>	Text	->	ファイルのPOSIX相対パス名
戻り値	4D.File	<-	<code>File</code> オブジェクト (無効なパスの場合には <code>null</code>)

説明

`.file()` 関数は、`Folder` オブジェクト内に `File` オブジェクトを作成し、その参照を返します。

`path` には、返すべきファイルの相対的パスを POSIX 形式で渡します。このパスは、親フォルダーを起点として評価されます。

戻り値

`File` オブジェクト (無効な *path* の場合には null)。

例題

```
var $myPDF : 4D.File  
$myPDF:=Folder(fk documents folder).file("Pictures/info.pdf")
```

.files()

▶履歴

`.files({ options : Integer }) : Collection`

引数	タイプ	説明
options	Integer	-> ファイルリストのオプション
戻り値	Collection	<- 子ファイルオブジェクトのコレクション

説明

`.files()` 関数は、フォルダーに格納されている `File` オブジェクトのコレクションを返します。

エイリアスまたはシンボリックリンクは解決されません。

*options*引数を渡さなかった場合はデフォルトで、フォルダーの第一階層にあるファイルのみがコレクションに返されます。これには非表示のファイルや、フォルダーも含まれます。*options* 引数に以下の定数を一つ以上渡すことで、このふるまいを変更することができます:

定数	値	説明
<code>fk recursive</code>	1	コレクションには、指定フォルダーとそのサブフォルダーのファイルが含まれます
<code>fk ignore invisible</code>	8	非表示設定のファイルは表示されません

戻り値

`File` オブジェクトのコレクション。

例題 1

データベースフォルダー内に非表示ファイルがないかどうかを調べます:

```
var $all; $noInvisible : Collection  
$all:=Folder(fk database folder).files()  
$noInvisible:=Folder(fk database folder).files(fk ignore invisible)  
If($all.length#$noInvisible.length)  
    ALERT("データベースフォルダーには非表示のファイルが存在します。")  
End if
```

例題 2

ドキュメントフォルダー内にある、非表示でないファイルをすべて取得します:

```
var $recursive : Collection  
$recursive:=Folder(fk documents folder).files(fk recursive+fk ignore invisible)
```

.folder()

▶履歴

.folder(*path* : Text) : 4D.Folder

引数	タイプ	説明
<i>path</i>	Text	-> ファイルのPOSIX相対パス名
戻り値	4D.Folder	<- 作成された Folder オブジェクト (無効な <i>path</i> の場合には null)

説明

.folder() 関数は、親の Folder オブジェクト内に新しい Folder オブジェクトを作成し、その参照を返します。

path には、返すべきフォルダーの相対的パスを POSIX 形式で渡します。このパスは、親フォルダーを起点として評価されます。

戻り値

Folder オブジェクト (無効な *path* の場合には null)。

例題

```
var $mypicts : 4D.Folder  
$mypicts:=Folder(fk documents folder).folder("Pictures")
```

.folders()

▶履歴

.folders({ *options* : Integer }) : Collection

引数	タイプ	説明
<i>options</i>	Integer	-> フォルダーリストのオプション
戻り値	Collection	<- 子フォルダーオブジェクトのコレクション

説明

.folders() 関数は、親フォルダーに格納されている Folder オブジェクトのコレクションを返します。

*options*引数を渡さなかった場合はデフォルトで、フォルダーの第一階層にあるフォルダーのみがコレクションに返されます。 *options* 引数に以下の定数を一つ以上渡すことで、このふるまいを変更することができます:

定数	値	説明
fk recursive	1	コレクションには、指定フォルダーとそのサブフォルダーのフォルダーが含まれます
fk ignore invisible	8	非表示設定のフォルダーは表示されません

戻り値

Folder オブジェクトのコレクション。

例題

データベースフォルダー内にあるすべてのフォルダーおよびサブフォルダーのコレクションを取得します:

```
var $allFolders : Collection  
$allFolders:=Folder("/PACKAGE").folders(fk recursive)
```

.fullName

▶ 覆歴

.fullName : Text

説明

.fullName プロパティは、拡張子 (あれば) を含めたフォルダーの完全な名称を返します。

このプロパティは 読み取り専用 です。

.getIcon()

▶ 覆歴

.getIcon({ size : Integer }) : Picture

引数	タイプ		説明
size	Integer	->	取得するピクチャーの一辺の長さ (ピクセル単位)
戻り値	Picture	<-	アイコン

説明

.getIcon() 関数は、フォルダーのアイコンを返します。

任意の size 引数を渡すと、返されるアイコンのサイズをピクセル単位で指定することができます。この値は、実際にはアイコンを格納している正方形の一辺の長さを表しています。アイコンは通常、32x32ピクセル ("大きいアイコン") または 16x16ピクセル ("小さいアイコン") で定義されています。この引数に 0 を渡すか省略した場合、"大きいアイコン" が返されます。

フォルダーがディスク上に存在しない場合、デフォルトの空のアイコンが返されます。

戻り値

フォルダーアイコンの [ピクチャー](#)。

.hidden

▶ 覆歴

.hidden : Boolean

説明

.hidden プロパティは、フォルダーがシステムレベルで "非表示" に設定されていれば trueを返します (それ以外の場合は false)。

このプロパティは 読み取り専用 です。

.isAlias

▶ 覆歴

.isAlias : Boolean

説明

.isAlias プロパティは、 Folder オブジェクトに対しては常に falseを返します。

このプロパティは 読み取り専用 です。

.isFile

▶ 覆歴

.isFile : Boolean

説明

`.isFile` プロパティは、フォルダーに対しては常に `false`を返します。

このプロパティは 読み取り専用 です。

.isFolder

▶ 履歴

`.isFolder` : Boolean

説明

`.isFolder` プロパティは、フォルダーに対しては常に `true`を返します。

このプロパティは 読み取り専用 です。

.isPackage

▶ 履歴

`.isPackage` : Boolean

説明

`.isPackage` プロパティは、フォルダーが macOS上のパッケージである (かつディスク上に存在している) 場合に `true`を返します。それ以外の場合には `false` を返します。

Windows 上においては、`.isPackage` は常に `false` を返します。

このプロパティは 読み取り専用 です。

.modificationDate

▶ 履歴

`.modificationDate` : Date

説明

`.modificationDate` プロパティは、フォルダーの最終変更日を返します。

このプロパティは 読み取り専用 です。

.modificationTime

▶ 履歴

`.modificationTime` : Time

説明

`.modificationTime` プロパティは、フォルダーの最終変更時刻 を返します (00:00 からの経過秒数の形式)。

このプロパティは 読み取り専用 です。

.moveTo()

▶ 履歴

`.moveTo(destinationFolder : 4D.Folder { ; newName : Text }) : 4D.Folder`

引数	タイプ		説明
destinationFolder	4D.Folder	->	宛先フォルダー
newName	Text	->	移動先でのフォルダーの完全な名称
戻り値	4D.Folder	<-	移動したフォルダー

説明

.moveTo() 関数は、`Folder` オブジェクト (ソースフォルダー) を `destinationFolder` が指定する移行先へと移動すると同時に、`newName` を指定した場合は名称も変更します。

`destinationFolder` 引数が指定するフォルダーはディスク上に存在している必要があります、そうでない場合にはエラーが生成されます。

デフォルトで、移動したフォルダーは元の名前を維持します。移動の際にフォルダー名を変更したい場合、新しい完全な名前を `newName` に渡します。新しい名前は命名規則に則っている必要があります (例: ":", "/", 等の文字を含んでいない、など)。そうでない場合、エラーが返されます。

返されるオブジェクト

移動後の `Folder` オブジェクト。

例題

フォルダーを移動し、名称も変更します:

```
var $tomeove; $moved : Object
$docs:=Folder(fk documents folder)
$tomeove:=$docs.folder("Pictures")
$tomeove2:=$tomeove.moveTo($docs.folder("Archives");"Pic_Archives")
```

.name

▶ 履歴

.name : Text

説明

.name プロパティは、拡張子 (あれば) を含まないフォルダー名を返します。

このプロパティは 読み取り専用 です。

.original

▶ 履歴

.original : 4D.Folder

説明

.original プロパティは、対象フォルダーと同じフォルダーオブジェクトを返します。

このプロパティは 読み取り専用 です。

このプロパティは、フォルダーやファイルを処理する汎用的なコードを書くために使用できます。

.parent

▶ 履歴

.parent : 4D.Folder

説明

.parent プロパティは、対象フォルダーの親フォルダーオブジェクトを返します。パスがシステムパスを表す場合 (例: "/DATA/")、システムパスが返されます。

親フォルダーが存在しない場合 (root) は、このプロパティは null 値を返します。

このプロパティは 読み取り専用 です。

.path

▶ 履歴

.path : Text

説明

.path プロパティは、フォルダーの POSIX パスを返します。パスがファイルシステムを表す場合 (例: "/DATA/")、ファイルシステムが返されます。

このプロパティは 読み取り専用 です。

.platformPath

▶ 履歴

.platformPath : Text

説明

.platformPath プロパティは、カレントプラットフォームのシンタックスで表現されたフォルダーのパスを返します。

このプロパティは 読み取り専用 です。

.rename()

▶ 履歴

.rename(newName : Text) : 4D.Folder

引数	タイプ		説明
newName	Text	->	フォルダーの新しい完全な名称
戻り値	4D.Folder	<-	名称変更されたフォルダー

説明

.rename() 関数は、フォルダー名を newName に指定した名称に変更し、名称変更後の Folder オブジェクトを返します。

newName 引数は命名規則に則っている必要があります (例: ":", "/", 等の文字を含んでいない、など)。そうでない場合、エラーが返されます。同じ名前のファイルがすでに存在する場合には、エラーが返されます。

返されるオブジェクト

名称変更された Folder オブジェクト。

例題

```
var $toRename : 4D.Folder  
$toRename:=Folder("/RESOURCES/Pictures").rename("Images")
```

Formula

Formula あるいは [Formula from string](#) コマンドを使用すると、ネイティブな [4D.Function オブジェクト](#) を作成することができ、それによってあらゆる 4D式やテキストとして表されたコードを実行することができます。

Formula オブジェクト

Formulaオブジェクトは、オブジェクトプロパティに格納することができます。

```
var $f : 4D.Function  
$f:=New object  
$f.message:=Formula(ALERT("Hello world"))
```

このようなプロパティは "オブジェクト関数"、つまり親オブジェクトに紐づいた関数です。オブジェクトプロパティに保存されている関数を実行するには、プロパティ名のあとに () をつけます：

```
$f.message() // "Hello world" を表示します
```

大カッコを使用したシンタックスもサポートされます：

```
$f["message"]() // "Hello world" と表示します
```

たとえ引数を受け取らなかつたとしても（後述参照）、オブジェクト関数を実行するためにはカッコ（）をつけて呼び出す必要があるという点に注意してください。オブジェクトプロパティのみを呼び出した場合、フォーミュラへの新しい参照が返されます（そしてフォーミュラは実行はされません）：

```
$o:=$f.message // $o にはフォーミュラオブジェクトが返されます
```

[apply\(\)](#) および [call\(\)](#) 関数を使って関数を実行することもできます：

```
$f.message.apply() // "Hello world!" を表示します
```

引数の受け渡し

フォーミュラには、[順番引数シンタックス](#) \$1, \$2...\$n を使用して引数を渡すことができます。たとえば：

```
var $f : Object  
$f:=New object  
$f.message:=Formula(ALERT("Hello "+$1))  
$f.message("John") // "Hello John" を表示します
```

あるいは、[.call\(\)](#) 関数を使用して：

```
var $f : Object  
$f:=Formula($1+" "+$2)  
$text:=$f.call(Null;"Hello";"World") // "Hello World" を返します  
$text:=$f.call(Null;"Welcome to";String(Year of(Current date))) // "Welcome to 2019" (例) を返します
```

单一メソッド用の引数

利便性のために、フォーミュラが単一のプロジェクトメソッドから作成された場合には、引数はフォーミュラオブジェクトの初期化では省略することができます。省略された引数は、フォーミュラを呼び出す時に一緒に渡すことができます。たとえば：

```
var $f : 4D.Function

$f:=Formula(myMethod)
// ここで Formula(myMethod($1;$2) と書く必要はありません
$text:=$f.call(Null;"Hello";"World") // "Hello World" を返します
$text:=$f.call() // "How are you?" を返します

//myMethod
#DECLARE ($param1 : Text; $param2 : Text)->$return : Text
If(Count parameters=2)
    $return:=$param1+" "+$param2
Else
    $return:="How are you?"
End if
```

引数はメソッド内において、呼び出し時に指定した順で受け取られます。

4D.Function オブジェクトについて

4D.Function オブジェクトにはコードが格納されています。このコードは () 演算子を使用して、または `apply()` や `call()` 関数を使用して呼び出すことができます。4D では 3種類の Function オブジェクトが利用できます：

- ネイティブ関数、つまり、`collection.sort()` や `file.copyTo()` などの 4Dクラスにビルトインされた関数。
- ユーザー関数 (ユーザークラスにおいて `Function` キーワードを使って作成されたもの)。
- フォーミュラ関数 (4Dフォーミュラを実行するもの)。

概要

`.apply() : any`
`.apply(thisObj : Object { ; formulaParams : Collection }) : any`
対象の `Formula` オブジェクトを実行し、その結果の値を返します

`.call() : any`
`.call(thisObj : Object { ; ...params : any }) : any`
対象の `Formula` オブジェクトを実行し、その結果の値を返します

`.source : Text`
対象フォーミュラのテキスト型のソース式

Formula

▶ 覆歴

`Formula (formulaExp : Expression) : 4D.Function`

引数	タイプ		説明
formulaExp	式	->	オブジェクトとして返されるフォーミュラ
戻り値	4D.Function	<-	フォーミュラを格納しているネイティブな Function オブジェクト

説明

`Formula` コマンドは、`formulaExp` の式に基づいた 4D Function オブジェクトを作成します。`formulaExp` には単一の値のようにシンプルなものから、引数を持つプロジェクトメソッドのように複雑なものまで指定することができます。

フォーミュラがオブジェクトとして存在することで、コマンドやメソッドに対して引数（計算された属性）として渡したり、"コンポーネントとホストデータベース間で共有"として宣言せよとも様々なコンポーネントから実行したりできるようになります。呼び出されたフォーミュラオブジェクトは、それを作成したデータベースあるいはコンポーネントのコンテキストにおいて評価されます。

返されたフォーミュラは以下の方法で呼び出すことが可能です：

- `.call()` あるいは `.apply()` 関数
- オブジェクト記法シンタックス ([Formula オブジェクト 参照](#))

```
var $f : 4D.Function
$f:=Formula(1+2)
$o:=New object("myFormula";$f)

// フォーミュラを呼び出す 3つの方法
$f.call($o) // 3 を返します
$f.apply($o) // 3 を返します
$o.myFormula() // 3 を返します
```

フォーミュラには [引数](#) を渡すことができます ([例題4 参照](#))。

フォーミュラの実行対象となるオブジェクトを指定することができます ([例題5 参照](#))。このオブジェクトのプロパティは、`This` コマンドでアクセス可能です。

`formulaExp` がローカル変数を使用する場合、返されるフォーミュラオブジェクトの作成時にその値がそこにコピーされ保存されます。実行時、フォーミュラはそのローカル変数の現在値ではなく、コピーされた値を使用します。ローカル変数として配列を使用することはサポートされていない点に注意してください。

`Formula` によって作成されたオブジェクトは、たとえばデータベースのフィールドや Blob ドキュメントなどに保存可能です。

例題 1

単純なフォーミュラの例：

```
var $f : 4D.Function
$f:=Formula(1+2)

var $o : Object
$o:=New object("f";$f)

$result:=$o.f() // 3 を返します
```

例題 2

ローカル変数を使用するフォーミュラの例：

```
$value:=10
$o:=New object("f";Formula($value))
$value:=20

$result:=$o.f() // 10 を返します
```

例題 3

引数を用いたシンプルなフォーミュラの例：

```
$o:=New object("f";Formula($1+$2))
$result:=$o.f(10;20) // 30 を返します
```

例題 4

引数を用いたプロジェクトメソッドを使用する例:

```
$o:=New object("f";Formula(myMethod))
$result:=$o.f("param1";"param2") // $result:=myMethod("param1";"param2") と同等です
```

例題 5

This を使用する例:

```
$o:=New object("fullName";Formula(This.firstName+" "+This.lastName))
$o.firstName:="John"
$o.lastName:="Smith"
$result:=$o.fullName() // "John Smith" を返します
```

例題 6

オブジェクト記法を使用してフォーミュラを呼び出す例:

```
var $feta; $robot : Object
var $calc : 4D.Function
$robot:=New object("name";"Robot";"price";543;"quantity";2)
$feta:=New object("name";"Feta";"price";12.5;"quantity";5)

$calc:=Formula(This.total:=This.price*This.quantity)

// フォーミュラをオブジェクトプロパティに設定します
$feta.calc:=$calc
$robot.calc:=$calc

// フォーミュラを呼び出します
$feta.calc() // $feta={name:Feta,price:12.5,quantity:5,total:62.5,calc:"[object Formula]"}
$robot.calc() // $robot={name:Robot,price:543,quantity:2,total:1086,calc:"[object Formula]"}
```

Formula from string

▶ 補足

Formula from string(*formulaString* : Text) : 4D.Function

引数	タイプ		説明
formulaString	Text	->	オブジェクトとして返されるフォーミュラ文字列
戻り値	4D.Function	<-	フォーミュラを格納しているネイティブなオブジェクト

説明

`Formula from string` コマンドは、*formulaString* に基づいた 4D Function オブジェクトを作成します。*formulaString* には単一の値のようにシンプルなものから、引数を持つプロジェクトメソッドのように複雑なものまで指定することができます。

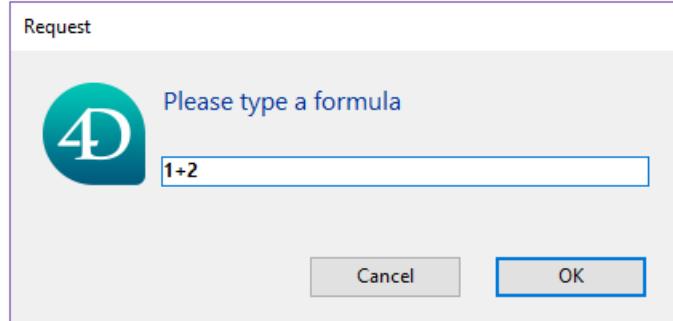
このコマンドは `Formula` に似ていますが、テキストに基づいたフォーミュラを扱う点が異なります。多くの場合において、`Formula` コマンドの使用が推奨されます。`Formula from string` コマンドは、元となるフォーミュラがテキストとして表現されている場合（例：外部の JSON ファイルに保存されていた場合など）にのみ使用されるべきです。このコンテキストにおいては、トークンシンタックスの使用が強く推奨されます。

ローカル変数の中身はコンパイル済みモードでは名前によるアクセスが不可能なため、*formulaString* 引数内で使用することはできません。`Formula from string` コマンドを使用してローカル変数にアクセスを試みた場合、エラー(-10737) が生成されます。

例題

以下のコードは、テキストフォーマットのフォーミュラを受け入れるダイアログを作成し、：

```
var $textFormula : Text
var $f : 4D.Function
$textFormula:=Request("フォーミュラを入力してください")
If(ok=1)
    $f:=Formula from string($textFormula)
    ALERT("結果 = "+String($f.call()))
End if
```



そのフォーミュラを実行します：



.apply()

▶履歴

.apply() : any
.apply(thisObj : Object { ; formulaParams : Collection }) : any

引数	タイプ		説明
thisObj	Object	->	フォーミュラ内で This コマンドによって返されるオブジェクト
formulaParams	Collection	->	フォーミュラが実行される際に \$1...\$n として渡される値のコレクション
戻り値	any	<-	フォーミュラの実行結果

説明

.apply() 関数は、対象の `Formula` オブジェクトを実行し、その結果の値を返します。 `Formula` あるいは `Formula from string` コマンドで作成されたフォーミュラが使用可能です。

`thisObj` には、フォーミュラ内で `This` として使用されるオブジェクトへの参照を渡すことができます。

任意の `formulaParams` 引数を渡すことで、フォーミュラ内で `$1...$n` の引数として使用されるコレクションを渡すこともできます。

`.apply()` は `.call()` と似ていますが、引数をコレクションとして渡す点が異なります。これは計算された結果を渡すのに便利です。

例題 1

```
var $f : 4D.Function  
$f:=Formula($1+$2+$3)  
  
$c:=New collection(10;20;30)  
$result:=$f.apply(NULL;$c) // 60 を返します
```

例題 2

```
var $calc : 4D.Function  
var $feta; $robot : Object  
$robot:=New object("name";"Robot";"price";543;"quantity";2)  
$feta:=New object("name";"Feta";"price";12.5;"quantity";5)  
  
$calc:=Formula(This.total:=This.price*This.quantity)  
  
$calc.apply($feta) // $feta={name:Feta,price:12.5,quantity:5,total:62.5}  
$calc.apply($robot) // $robot={name:Robot,price:543,quantity:2,total:1086}
```

.call()

▶ 覆歴

`.call()` : any
`.call(thisObj : Object { ; ...params : any })` : any

引数	タイプ		説明
thisObj	Object	->	フォーミュラ内で <code>This</code> コマンドによって返されるオブジェクト
params	any	->	フォーミュラが実行される際に <code>\$1...\$n</code> として渡される値
戻り値	any	<-	フォーミュラの実行結果

説明

`.call()` 関数は、対象の `Formula` オブジェクトを実行し、その結果の値を返します。`Formula` あるいは `Formula from string` コマンドで作成されたフォーミュラが使用可能です。

`thisObj` には、フォーミュラ内で `This` として使用されるオブジェクトへの参照を渡すことができます。

任意の `params` 引数を渡すことで、フォーミュラ内で `$1...$n` の引数として使用される値を渡すこともできます。

`.call()` は `.apply()` と似ていますが、引数を直接渡す点が異なります。

例題 1

```
var $f : 4D.Function  
$f:=Formula(Uppercase($1))  
$result:=$f.call(NULL;"hello") // "HELLO" を返します
```

例題 2

```
$o:=New object("value";50)
$f:=Formula(This.value*2)
$result:=$f.call($o) // 100 を返します
```

.source

▶ 覆歴

.source : Text

説明

.source プロパティは、対象フォーミュラのテキスト型のソース式を格納します。

このプロパティは 読み取り専用 です。

例題

```
var $of : 4D.Function
var $tf : Text
$of:=Formula(String(Current time;HH MM AM PM))
$tf:=$of.source //"String(Current time;HH MM AM PM)"
```

IMAPTransporter

`IMAPTransporter` クラスを使って、IMAP メールサーバーからメッセージを取得することができます。

IMAP Transporter オブジェクト

IMAP Transporter オブジェクトは [IMP New transporter](#) コマンドによってインスタンス化されます。これらは、次のプロパティや関数を持ちます：

`.acceptUnsecureConnection : Boolean`

暗号化されていない接続の確立が許可されれば `true`

`.addFlags(msgIDs : Collection ; keywords : Object) : Object`

`.addFlags(msgIDs : Text ; keywords : Object) : Object`

`.addFlags(msgIDs : Longint ; keywords : Object) : Object`

`msgIDs` のメッセージに対して、`keywords` で指定したフラグを追加します

`.append(mailObj : Object ; destinationBox : Text ; options : Object) : Object`

`destinationBox` に指定したメールボックスに、`mailObj` のメールを追加します

`.authenticationMode : Text`

メールサーバーのセッションを開くのに使用される認証モード

`.checkConnection() : Object`

transporter オブジェクトが保存する情報を使用して接続をチェックします

`.checkConnectionDelay : Integer`

サーバー接続をチェックするまでの最長時間 (秒単位)

`.connectionTimeOut : Integer`

サーバー接続の確立までに待機する最長時間 (秒単位)

`.copy(msgsIDs : Collection ; destinationBox : Text) : Object`

`.copy(allMsgs : Integer ; destinationBox : Text) : Object`

`msgsIDs` または `allMsgs` で定義されたメッセージを IMAP サーバーの `destinationBox` へとコピーします

`.createBox(name : Text) : Object`

`name` に指定した名称の新規メールボックスを作成します

`.delete(msgsIDs : Collection) : Object`

`.delete(allMsgs : Integer) : Object`

`msgsIDs` または `allMsgs` が指定するメッセージに対して "削除済み" フラグを設定します

`.deleteBox(name : Text) : Object`

`name` に指定した名称のメールボックスを IMAP サーバーから完全に削除します

`.expunge() : Object`

"deleted" フラグがつけられたメッセージをすべて IMAP メールサーバーから削除します

`.getBoxInfo({ name : Text }) : Object`

カレントメールボックス、または `name` が指定するメールボックスに対応する `boxInfo` オブジェクトを返します

`.getBoxList({ parameters : Object }) : Collection`

利用可能なメールボックスの情報を `mailbox` オブジェクトのコレクションとして返します

`.getDelimiter() : Text`

メールボックス名で階層レベルを区切るのに使用される文字を返します

`.getMail(msgNumber: Integer { ; options : Object }) : Object`

`.getMail(msgID: Text { ; options : Object }) : Object`

`IMAP_transporter` が指定するメールボックス内の、`msgNumber` または `msgID` に対応するメールを `Email` オブジェクトとして返します

`.getMails(ids : Collection { ; options : Object }) : Object`

`.getMails(startMsg : Integer ; endMsg : Integer { ; options : Object }) : Object`

`Email` オブジェクトのコレクションを格納したオブジェクトを返します

`.getMIMEAsBlob(msgNumber : Integer { ; updateSeen : Boolean }) : Blob`

`.getMIMEAsBlob(msgID : Text { ; updateSeen : Boolean }) : Blob`

`IMAP_transporter` が指定するメールボックス内の、`msgNumber` または `msgID` に対応するメッセージの MIMEコンテンツを格納した BLOB を返します

`.host : Text`

ホストサーバーの名前または IPアドレス

`.logFile : Text`

メール接続に対して定義された拡張ログファイル (あれば) へのフルパス

`.move(msgsIDs : Collection ; destinationBox : Text) : Object`

`.move(allMsgs : Integer ; destinationBox : Text) : Object`

`msgsIDs` または `allMsgs` で定義されたメッセージを IMAP サーバーの `destinationBox` へと移動します

`.numToID(startMsg : Integer ; endMsg : Integer) : Collection`

`startMsg` および `endMsg` で指定された連続した範囲のメッセージのシーケンス番号を IMAP固有IDへと変換します

`.removeFlags(msgIDs : Collection ; keywords : Object) : Object`

`.removeFlags(msgIDs : Text ; keywords : Object) : Object`

`.removeFlags(msgIDs : Longint ; keywords : Object) : Object`

`msgIDs` のメッセージに対して、`keywords` で指定したフラグを削除します

`.renameBox(currentName : Text ; newName : Text) : Object`

IMAPサーバー上でメールボックスの名称を変更します

`.port : Integer`

メール通信に使用されるポート番号

`.searchMails(searchCriteria : Text) : Collection`

カレントメールボックスにおいて `searchCriteria` の検索条件に合致するメッセージを検索します

`.selectBox(name : Text { ; state : Integer }) : Object`

`name` に指定したメールボックスをカレントメールボックスとして選択します

`.subscribe(name : Text) : Object`

IMAPサーバーの購読メールボックスとして任意のメールボックスを追加・削除します

`.unsubscribe(name : Text) : Object`

指定したメールボックスを購読メールボックスから削除します

`.user : Text`

メールサーバーでの認証に使用されたユーザー名

IMAP New transporter

▶履歴

IMAP New transporter(*server* : Object) : 4D.IMAPTransporter

引数	タイプ		説明
<i>server</i>	Object	->	メールサーバー情報
戻り値	4D.IMAPTransporter	<-	IMAP transporter オブジェクト

説明

`IMAP New transporter` コマンドは、*server* 引数の指定に応じて 新規の IMAP接続を設定します。戻り値は、新しい *IMAP transporter* オブジェクトです。返された transporter オブジェクトは、通常メールの受信に使用されます。

server 引数として、以下のプロパティを持つオブジェクトを渡します:

<i>server</i>		デフォルト値 (省略時)
.acceptUnsecureConnection : Boolean 暗号化されていない接続の確立が許可されてれば true	False	
.accessTokenOAuth2: Text .accessTokenOAuth2: Object OAuth2 認証の資格情報を表すテキスト文字列またはトークンオブジェクト。 <i>authenticationMode</i> が OAUTH2 の場合のみ使用されます。 <i>accessTokenOAuth2</i> が使用されているが <i>authenticationMode</i> が省略されていた場合、OAuth2 プロトコルが使用されます (サーバーで許可されていれば)。 <i>IMAP transporter</i> オブジェクトには返されません。	なし	
.authenticationMode : Text メールサーバーのセッションを開くのに使用される認証モード	サーバーがサポートするもっともセキュアな認証モードが使用されます	
.checkConnectionDelay : Integer サーバー接続をチェックするまでの最長時間 (秒単位)	300	
.connectionTimeOut : Integer サーバー接続の確立までに待機する最長時間 (秒単位)	30	
.host : Text ホストサーバーの名前または IPアドレス	必須	
.logFile : Text メール接続に対して定義された拡張ログファイル (あれば) へのフルパス	なし	
.password : Text サーバーとの認証のためのユーザーパスワード <i>IMAP transporter</i> オブジェクトには返されません。	なし	
.port : Integer メール通信に使用されるポート番号	993	
.user : Text メールサーバーでの認証に使用されたユーザー名	なし	

警告: 定義されたタイムアウトが、サーバータイムアウトより短いようにしてください。そうでない場合、クライアントタイムアウトは無意味になります。

戻り値

この関数は、IMAP transporter オブジェクト を返します。返されるプロパティはすべて 読み取り専用 です。

IMAP接続は、transporter オブジェクトが消去された時点で自動的に閉じられます。

例題

```
$server:=New object
$server.host:="imap.gmail.com" // 必須
$server.port:=993
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXX"
$server.logFile:="LogTest.txt" // Logsフォルダーに保存するログファイル

var $transporter : 4D.IMAPTransporter
$transporter:=IMAP New transporter($server)

$status:=$transporter.checkConnection()
If(Not($status.success))
    ALERT("エラーが発生しました: "+$status.statusText)
End if
```

4D.IMAPTransporter.new()

4D.IMAPTransporter.new(server : Object) : 4D.IMAPTransporter

引数	タイプ		説明
server	Object	->	メールサーバー情報
戻り値	4D.IMAPTransporter	<-	IMAP transporter オブジェクト

説明

4D.IMAPTransporter.new() 関数は、新規の 4D.IMAPTransporter 型オブジェクトを作成して返します。この関数の機能は、IMAP New transporter コマンドと同一です。

.acceptUnsecureConnection

▶ 履歴

.acceptUnsecureConnection : Boolean

説明

.acceptUnsecureConnection プロパティは、暗号化された接続が不可能な場合に、暗号化されていない接続の確立が許可されてれば true を格納します。

暗号化されていない接続が許可されていない場合には false が格納されており、その場合に暗号化された接続が不可能な場合にはエラーが返されます。

使用可能なセキュアなポートは次のとおりです:

- SMTP
 - 465: SMTPS
 - 587 または 25: STARTTLS アップグレードがされた SMTP (サーバーがサポートしていれば)
- IMAP
 - 143: IMAP 非暗号化ポート

- 993: STARTTLS アップグレードがされた IMAP (サーバーがサポートしていれば)
- POP3
 - 110: POP3 非暗号化ポート
 - 995: STARTTLS アップグレードがされた POP3 (サーバーがサポートしていれば)

.addFlags()

▶ 履歴

.addFlags(msgIDs : Collection ; keywords : Object) : Object

.addFlags(msgIDs : Text ; keywords : Object) : Object

.addFlags(msgIDs : Longint ; keywords : Object) : Object

引数	タイプ		説明
msgIDs	Collection	->	文字列のコレクション: メッセージの固有ID (テキスト型) テキスト: メッセージの固有ID 倍長整数 (IMAP all): 選択されたメールボックス内の全メッセージ
keywords	Object	->	追加するキーワードフラグ
戻り値	Object	<-	addFlags処理のステータス

説明

.addFlags() 関数は、 msgIDs のメッセージに対して、 keywords で指定したフラグを追加します。

msgIDs には、以下のいずれかを渡すことができます:

- 指定するメッセージの固有ID を格納した コレクション
- 単一のメッセージの固有ID (テキスト)
- 以下の定数 (longint) を使用することで、選択されているメールボックスの全メッセージを指定することができます:

定数	値	説明
IMAP all	1	選択されたメールボックスの全メッセージを選択します

keywords には、 msgIDs 引数で指定したメッセージに対して追加するフラグのキーワード値を格納したオブジェクトを渡します。次のキーワードを渡すことができます:

引数	タイプ	説明
\$draft	Boolean	メッセージに "draft" フラグを追加するには true
\$seen	Boolean	メッセージに "seen" フラグを追加するには true
\$flagged	Boolean	True to add the "flagged" flag to the message
\$answered	Boolean	メッセージに "answered" フラグを追加するには true
\$deleted	Boolean	メッセージに "deleted" フラグを追加するには true

- false値は無視されます。
- キーワードフラグの解釈は、メールクライアントごとに異なる可能性があります。

返されるオブジェクト

この関数は、IMAP ステータスを表すオブジェクトを返します:

プロパティ		タイプ	説明
success		Boolean	処理が正常に終わった場合には true、それ以外は false
statusText		Text	IMAPサーバーから返されたステータスマッセージ、または 4Dエラースタック内に返された最後のエラー
errors		Collection	4Dエラースタック (IMAPサーバーレスポンスが受信できた場合には返されません)
	[].errcode	Number	4Dエラーコード
	[].message	Text	4Dエラーの詳細
	[].componentSignature	Text	エラーを返した内部コンポーネントの署名

例題

```
var $options;$transporter;$boxInfo;$status : Object

$options:=New object
$options.host:="imap.gmail.com"
$options.port:=993
$options.user:="4d@gmail.com"
$options.password:="xxxxx"

// transporter を作成します
$transporter:=IMAP New transporter($options)

// メールボックスを選択します
$boxInfo:=$transporter.selectBox("INBOX")

// INBOXの全メッセージを既読に設定します
$flags:=New object
$flags["$seen"]:=True
$status:=$transporter.addFlags(IMAP all;$flags)
```

.append()

▶履歴

.append(*mailObj* : Object ; *destinationBox* : Text ; *options* : Object) : Object

引数	タイプ		説明
<i>mailObj</i>	Object	->	Email オブジェクト
<i>destinationBox</i>	Text	->	Emailオブジェクトを受信するメールボックス
<i>options</i>	Object	->	文字セット情報を格納したオブジェクト
戻り値	Object	<-	append処理のステータス

説明

.append() 関数は、 destinationBox に指定したメールボックスに、 mailObj のメールを追加します。

mailObj には、Email オブジェクトを渡します。メールプロパティに関する包括的な詳細については、Email オブジェクト を参照ください。

.append() 関数は Email オブジェクトの keywords 属性内のキーワードタグをサポートします。

任意の destinationBox には、 mailObj が追加されるメールボックスの名前を指定することができます。省略した場合は、カレントメールボックスが使用されます。

任意の options には、メールの特定部分の文字セットやエンコーディングを定義するオブジェクトを渡すことができます。次のプロパティを含みます：

プロパティ	タイプ	説明
headerCharset	Text	メールの以下の部分で使用される文字セットとエンコーディング: 件名、添付ファイル名、メール名の属性。取り得る値: 以下の可能な文字セットテーブルを参照ください。
bodyCharset	Text	メールの HTML およびテキスト本文コンテンツで使用される文字セットとエンコーディング。取り得る値: 以下の可能な文字セットテーブルを参照ください。

使用可能な文字セット:

定数	値	説明
mail mode ISO2022JP	US-ASCII_ISO-2022-JP_UTF8_QP	<ul style="list-style-type: none"> headerCharset: 可能なら US-ASCII、次に可能なら Japanese (ISO-2022-JP) & Quoted-printable、それも不可なら UTF-8 & Quoted-printable bodyCharset: 可能なら US-ASCII、次に可能なら Japanese (ISO-2022-JP) & 7-bit、それも不可なら UTF-8 & Quoted-printable
mail mode ISO88591	ISO-8859-1	<ul style="list-style-type: none"> headerCharset: ISO-8859-1 & Quoted-printable bodyCharset: ISO-8859-1 & 8-bit
mail mode UTF8	US-ASCII_UTF8_QP	headerCharset & bodyCharset: 可能なら US-ASCII、それが不可なら UTF-8 & Quoted-printable (デフォルト値)
mail mode UTF8 in base64	US-ASCII_UTF8_B64	headerCharset & bodyCharset: 可能な場合は US-ASCII、それ以外は UTF-8 & base64

返されるオブジェクト

この関数は、IMAP ステータスを表すオブジェクトを返します:

プロパティ		タイプ	説明
success		Boolean	処理が正常に終わった場合には true、それ以外は false
statusText		Text	IMAPサーバーから返されたステータスマッセージ、または 4Dエラースタック内に返された最後のエラー
errors		Collection	4Dエラースタック (IMAPサーバーレスポンスが受信できた場合には返されません)
	[].errcode	Number	4Dエラーコード
	[].message	Text	4Dエラーの詳細
	[].componentSignature	Text	エラーを返した内部コンポーネントの署名

例題

Drafts メールボックスにメールを保存します:

```

var $settings; $status; $msg; $imap: Object

$settings:=New object("host"; "domain.com"; "user"; "xxxx"; "password"; "xxxx"; "port"; 993)

$imap:=IMAP New transporter($settings)

$msg:=New object
$msg.from:="xxxx@domain.com"
$msg.subject:="Lorem Ipsum"
$msg.textBody:="Lorem ipsum dolor sit amet, consectetur adipiscing elit."
$msg.keywords:=New object
$msg.keywords["$seen"]:=True // メッセージに既読フラグをつきます
$msg.keywords["$draft"]:=True // // メッセージに下書きフラグをつきます

$status:=$imap.append($msg; "Drafts")

```

.authenticationMode

▶ 補足

.authenticationMode : Text

説明

.authenticationMode プロパティは、メールサーバーのセッションを開くのに使用される認証モードを格納します。

デフォルトでは、サーバーによってサポートされている最も安全なモードが使用されます。

とりうる値:

値	定数	説明
CRAM-MD5	IMAP authentication CRAM MD5	CRAM-MD5 プロトコルを使用した認証
LOGIN	IMAP authentication login	LOGIN プロトコルを使用した認証
OAuth2	IMAP authentication OAuth2	OAuth2 プロトコルを使用した認証
PLAIN	IMAP authentication plain	PLAIN プロトコルを使用した認証

.checkConnection()

▶ 補足

.checkConnection() : Object

引数	タイプ	説明
戻り値	Object	<- transporter オブジェクト接続のステータス

説明

.checkConnection() 関数は、transporter オブジェクトが保存する情報を使用して接続をチェックします。必要なら再接続をし、そのステータスを返します。この関数を使用して、ユーザーから提供された値が有効かどうかを検証することができます。

返されるオブジェクト

この関数はメールサーバーにリクエストを送信し、メールステータスを表すオブジェクトを返します。このオブジェクトには、次のプロパティが格納されることがあります:

プロパティ		タイプ	説明
success		boolean	チェックが成功した場合には true、それ以外は false
status		number	(SMTPのみ) メールサーバーから返されたコード (メール処理に関係ない問題の場合には 0)
statusText		テキスト	メールサーバーから返されたステータスマッセージ、または 4Dエラースタック内に返された最後のエラー
errors		collection	4Dエラースタック (メールサーバーレスポンスが受信できた場合には返されません)
[].errCode		number	4Dエラーコード
[].message		テキスト	4Dエラーの詳細
[].componentSignature		テキスト	エラーを返した内部コンポーネントの署名

.checkConnectionDelay

▶履歴

.checkConnectionDelay : Integer

説明

.checkConnectionDelay 関数は、サーバー接続をチェックするまでの最長時間 (秒単位)を格納します。関数呼び出しの間隔がこの時間を超過する場合、サーバー接続が確認されます。プロパティが server オブジェクトによって設定されていない場合は、デフォルトで 300 という値が使用されます。

警告: 定義されたタイムアウトが、サーバータイムアウトより短いようにしてください。そうでない場合、クライアントタイムアウトは無意味になります。

.connectionTimeOut

▶履歴

.connectionTimeOut : Integer

説明

.connectionTimeOut プロパティは、サーバー接続の確立までに待機する最長時間 (秒単位)を格納します。 SMTP New transporter や POP3 New transporter 、 IMAP New transporter のコマンドで transporter オブジェクトを作成する際に使用される server オブジェクトにおいて、このプロパティが指定されなかった場合のデフォルトは 30 です。

.copy()

▶履歴

.copy(msgsIDs : Collection ; destinationBox : Text) : Object

.copy(allMsgs : Integer ; destinationBox : Text) : Object

引数	タイプ		説明
msgsIDs	Collection	->	メッセージの固有ID のコレクション (テキスト)
allMsgs	Integer	->	IMAP all : 選択されたメールボックスの全メッセージ
destinationBox	Text	->	メッセージのコピー先のメールボックス
戻り値	Object	<-	copy処理のステータス

説明

.copy() 関数は、 msgsIDs または allMsgs で定義されたメッセージを IMAP サーバーの destinationBox へとコピーします。

以下のものを渡すことができます:

- *msgIDs* には、コピーするメッセージの固有ID を格納したコレクション
- *allMsgs* には、選択されているメールボックスの全メッセージをコピーするための定数 (倍長整数型):

destinationBox には、メッセージのコピー先メールボックスの名称をテキスト値で渡すことができます。

返されるオブジェクト

この関数は、IMAP ステータスを表すオブジェクトを返します:

プロパティ		タイプ	説明
success		Boolean	処理が正常に終わった場合には true、それ以外は false
statusText		Text	IMAPサーバーから返されたステータスマッセージ、または 4Dエラースタック内に返された最後のエラー
errors		Collection	4Dエラースタック (IMAPサーバーレスポンスが受信できた場合には返されません)
	[].errcode	Number	4Dエラーコード
	[].message	Text	4Dエラーの詳細
	[].componentSignature	Text	エラーを返した内部コンポーネントの署名

例題 1

選択されたメッセージをコピーします:

```
var $server;$boxInfo;$status : Object
var $mailIds : Collection
var $transporter : 4D.IMAPTransporter

$server:=New object
$server.host:="imap.gmail.com" // 必須
$server.port:=993
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXX"

$transporter:=IMAP New transporter($server)

// メールボックスを選択します
$boxInfo:=$transporter.selectBox("inbox")

// メッセージの固有ID のコレクションを取得します
$mailIds:=$transporter.searchMails("subject \"4D new feature:\"")

// 見つかったメッセージを "documents" メールボックスへコピーします
$status:=$transporter.copy($mailIds;"documents")
```

例題 2

カレントメールボックスの全メッセージをコピーします:

```

var $server;$boxInfo;$status : Object
var $transporter : 4D.IMAPTransporter

$server:=New object
$server.host:="imap.gmail.com" // 必須
$server.port:=993
$server.user:="4d@gmail.com"
$server.password:="XXXXXXX"

$transporter:=IMAP New transporter($server)

// メールボックスを選択します
$boxInfo:=$transporter.selectBox("inbox")

// 全メッセージを "documents" メールボックスへコピーします
$status:=$transporter.copy(IMAP all;"documents")

```

.createBox()

▶ 覆歴

.createBox(name : Text) : Object

引数	タイプ		説明
name	Text	->	新規メールボックスの名称
戻り値	Object	<-	createBox処理のステータス

説明

.createBox() 関数は、 name に指定した名称の新規メールボックスを作成します。IMAPサーバーの階層区切り文字がメールボックス名内に含まれる場合、IMAPサーバーは指定のメールボックスを作成するのに必要な親階層を作成します。

たとえば、"/" が階層区切り文字として使われるサーバーにおいて、"Projects/IMAP/Doc" を作成しようとした場合:

- "Projects" & "IMAP" がすでに存在する場合は "Doc" メールボックスのみを作成します。
- "Projects" のみが存在する場合は、"IMAP" & "Doc" メールボックスを作成します。
- いずれも存在しない場合には、"Projects" & "IMAP" & "Doc" メールボックスをそれぞれ作成します。

name には、新しいメールボックスの名前を渡します。

返されるオブジェクト

この関数は、IMAP ステータスを表すオブジェクトを返します:

プロパティ		タイプ	説明
success		Boolean	処理が正常に終わった場合には true、それ以外は false
statusText		Text	IMAPサーバーから返されたステータスマッセージ、または 4Dエラースタック内に返された最後のエラー
errors		Collection	4Dエラースタック (IMAPサーバーレスポンスが受信できた場合には返されません)
	[].errcode	Number	4Dエラーコード
	[].message	Text	4Dエラーの詳細
	[].componentSignature	Text	エラーを返した内部コンポーネントの署名

例題

新しい "Invoices" メールボックスを作成します:

```

var $pw : text
var $options; $transporter; $status : object

$options:=New object

$pw:=Request("パスワードを入力してください:")
If(OK=1)
$options.host:="imap.gmail.com"
$options.user:="test@gmail.com"
$options.password:=$pw

$transporter:=IMAP New transporter($options)

$status:=$transporter.createBox("Invoices")

If ($status.success)
ALERT("メールボックスが作成できました。")
Else
ALERT("エラー: "+$status.statusText)
End if
End if

```

.delete()

▶ 覆歴

.delete(`msgsIDs` : Collection) : Object
 .delete(`allMsgs` : Integer) : Object

引数	タイプ		説明
<code>msgsIDs</code>	Collection	->	メッセージの固有ID のコレクション (テキスト)
<code>allMsgs</code>	Integer	->	IMAP all : 選択されたメールボックスの全メッセージ
戻り値	Object	<-	delete処理のステータス

説明

`.delete()` 関数は、`msgsIDs` または `allMsgs` が指定するメッセージに対して "削除済み" フラグを設定します。

以下のものを渡すことができます:

- `msgsIDs` には、削除するメッセージの固有ID を格納したコレクション
- `allMsgs` には、選択されているメールボックスの全メッセージを削除するための定数 (倍長整数型):

この関数を実行しても、メールが実際に削除される訳ではありません。"削除済み" フラグがつけられたメッセージも引き続き `.searchMails()` 関数によって検索可能です。フラグがつけられたメッセージは、`.expunge()` を実行したときか、別のメールボックスを選択したとき、あるいは(`IMAP New transporter` で作成された) `transporter オブジェクト` が消去されたときにのみ、IMAPサーバーから削除されます。

返されるオブジェクト

この関数は、IMAP ステータスを表すオブジェクトを返します:

プロパティ		タイプ	説明
success		Boolean	処理が正常に終わった場合には true、それ以外は false
statusText		Text	IMAPサーバーから返されたステータスマッセージ、または 4Dエラースタック内に返された最後のエラー
errors		Collection	4Dエラースタック (IMAPサーバーレスポンスが受信できた場合には返されません)
	[].errcode	Number	4Dエラーコード
	[].message	Text	4Dエラーの詳細
	[].componentSignature	Text	エラーを返した内部コンポーネントの署名

例題 1

選択されたメッセージを削除します:

```

var $server;$boxInfo;$status : Object
var $mailIds : Collection
var $transporter : 4D.IMAPTransporter

$server:=New object
$server.host:="imap.gmail.com" // 必須
$server.port:=993
$server.user:="4d@gmail.com"
$server.password:="XXXXXXX"

$transporter:=IMAP New transporter($server)

// メールボックスを選択します
$boxInfo:=$transporter.selectBox("Inbox")

// メッセージの固有ID のコレクションを取得します
$mailIds:=$transporter.searchMails("subject \"Reports\"")

// 選択されたメッセージを削除します
$status:=$transporter.delete($mailIds)

```

例題 2

カレントメールボックスの全メッセージを削除します:

```

var $server;$boxInfo;$status : Object
var $transporter : 4D.IMAPTransporter

$server:=New object
$server.host:="imap.gmail.com" // 必須
$server.port:=993
$server.user:="4d@gmail.com"
$server.password:="XXXXXXX"

$transporter:=IMAP New transporter($server)

//メールボックスを選択します
$boxInfo:=$transporter.selectBox("Junk Email")

// カレントメールボックスの全メッセージを削除します
$status:=$transporter.delete(IMAP all)

```

.deleteBox()

▶履歴

.deleteBox(*name* : Text) : Object

引数	タイプ	説明
<i>name</i>	Text	-> 削除するメールボックスの名称

|Result|Object|<-|deleteBox処理のステータス|

説明

.deleteBox() 関数は、 *name* に指定した名称のメールボックスを IMAPサーバーから完全に削除します。存在しないメールボックス、または INBOX を削除しようとして場合には、エラーが生成されます。

name には、削除するメールボックスの名前を渡します。

- 子メールボックスを持つ親メールボックスが "Noselect" 属性を持っている場合、そのメールボックスは削除できません。
- 削除されるメールボックス内のメッセージもすべて削除されます。
- メールボックス削除の可否はメールサーバーに依存します。

返されるオブジェクト

この関数は、IMAP ステータスを表すオブジェクトを返します:

プロパティ		タイプ	説明
success		Boolean	処理が正常に終わった場合には true、それ以外は false
statusText		Text	IMAPサーバーから返されたステータスマッセージ、または 4Dエラースタック内に返された最後のエラー
errors		Collection	4Dエラースタック (IMAPサーバーレスポンスが受信できた場合には返されません)
	[].errcode	Number	4Dエラーコード
	[].message	Text	4Dエラーの詳細
	[].componentSignature	Text	エラーを返した内部コンポーネントの署名

例題

"Bills" メールボックスの階層から、"Nova Orion Industries" の子メールボックスを削除します:

```

var $pw; $name : text
var $options; $transporter; $status : object

$options:=New object

$pw:=Request("パスワードを入力してください:")

If(OK=1) $options.host:="imap.gmail.com"
$options.user:="test@gmail.com"
$options.password:=$pw

$transporter:=IMAP New transporter($options)

// delete mailbox
$name:="Bills"+$transporter.getDelimiter()+"Nova Orion Industries"
$status:=$transporter.deleteBox($name)

If ($status.success)
    ALERT("メールボックスが削除されました。")
Else
    ALERT("エラー: "+$status.statusText)
End if
End if

```

.expunge()

▶ 覆歴

.expunge() : Object

引数	タイプ		説明
戻り値	オブジェクト	<-	expunge処理のステータス

説明

.expunge() 関数は、"deleted" フラグがつけられたメッセージをすべてIMAP メールサーバーから削除します。"deleted" フラグは [.delete\(\)](#) または [.addFlags\(\)](#) 関数によって設定可能です。

返されるオブジェクト

この関数は、IMAP ステータスを表すオブジェクトを返します：

プロパティ		タイプ	説明
success		Boolean	処理が正常に終わった場合には true、それ以外は false
statusText		Text	IMAPサーバーから返されたステータスマッセージ、または 4Dエラースタック内に返された最後のエラー
errors		Collection	4Dエラースタック (IMAPサーバーレスポンスが受信できた場合には返されません)
	[].errcode	Number	4Dエラーコード
	[].message	Text	4Dエラーの詳細
	[].componentSignature	Text	エラーを返した内部コンポーネントの署名

例題

```

var $options;$transporter;$boxInfo;$status : Object
var $ids : Collection

$options:=New object
$options.host:="imap.gmail.com"
$options.port:=993
$options.user:="4d@gmail.com"
$options.password:="xxxxx"

// transporter を作成します
$transporter:=IMAP New transporter($options)

// メールボックスを選択します
$boxInfo:=$transporter.selectBox("INBOX")

// INBOX の既読メッセージに削除フラグを立てます
$ids:=$transporter.searchMails("SEEN")
$status:=$transporter.delete($ids)

// "deleted" フラグがついたメッセージをすべて消去します
$status:=$transporter.expunge()

```

.getBoxInfo()

▶履歴

.getBoxInfo({ name : Text }) : Object

引数	タイプ		説明
name	テキスト	->	メールボックスの名称
戻り値	オブジェクト	<-	boxInfo オブジェクト

説明

.getBoxInfo() 関数は、カレントメールボックス、または name が指定するメールボックスに対応する boxInfo オブジェクトを返します。この関数は、.selectBox() と同じ情報を返しますが、カレントメールボックスは変えません。

任意の name パラメーターには、アクセスするメールボックスの名称を渡します。この名称は明確な左から右への階層を表し、特定の区切り文字でレベルを区分けします。この区切り文字は .getDelimiter() 関数で調べることができます。

name のメールボックスが選択不可の場合、または見つからない場合には、関数はエラーを生成し、null を返します。

返されるオブジェクト

返される boxInfo オブジェクトには、以下のプロパティが格納されています:

プロパティ	タイプ	説明
name	テキスト	メールボックスの名称
mailCount	number	メールボックス内のメッセージの数
mailRecent	number	(新しいメッセージであることを表す) "recent" フラグがついたメッセージの数

例題

```

var $transporter : 4D.IMAPTransporter
$transporter:=IMAP New transporter($server)

$info:=$transporter.getBoxInfo("INBOX")
ALERT("INBOX には "+String($info.mailRecent)+" 件の最近のメールがあります。")

```

.getBoxList()

▶履歴

.getBoxList({ parameters : Object }) : Collection

引数	タイプ		説明
parameters	オブジェクト	->	引数のオブジェクト
戻り値	コレクション	<-	mailbox オブジェクトのコレクション

説明

.getBoxList() 関数は、利用可能なメールボックスの情報を mailbox オブジェクトのコレクションとして返します。この関数を使用すると、IMAP メールサーバー上にあるメッセージの一覧をローカルで管理することができるようになります。

任意の parameters パラメーターには、返されるメールボックスをフィルターするための値を格納したオブジェクトを渡すことができます。以下のものを渡すことができます:

プロパティ	タイプ	説明
isSubscribed	Boolean	<ul style="list-style-type: none"> True: 購読しているメールボックスのみを返します。 False: すべての利用可能なメールボックスを返します

戻り値

返されるコレクションの各オブジェクトには、以下のプロパティが格納されています:

プロパティ	タイプ	説明
[] .name	テキスト	メールボックスの名称
[] .selectable	boolean	アクセス権でメールボックスを選択できるかどうかを表します: <ul style="list-style-type: none"> true - メールボックスは選択可能 false - メールボックスは選択不可能
[] .inferior	boolean	アクセス権でメールボックス内に下の階層レベルを作成できるかどうかを表します: <ul style="list-style-type: none"> true - 下の階層レベルは作成可能 false - 下の階層レベルは作成不可能
[] .interesting	boolean	サーバーがメールボックスに "interesting" のマーク付けをしているかどうかを表します: <ul style="list-style-type: none"> true - メールボックスはサーバーから "interesting" のマーク付けをされています。たとえば、メールボックスには新着メッセージが入っている場合が考えられます。 false - メールボックスはサーバーから "interesting" のマーク付けをされていません。

アカウントにメールボックスが一つもない場合、空のコレクションが返されます。

- 開いている接続がない場合、.getBoxList() は接続を開きます。
- 接続が指定された時間 (IMAP New transporter 参照) 以上に使用されなかった場合には、.checkConnection() 関数が自動的に呼び出されます。

例題

```
var $transporter : 4D.IMAPTransporter
$transporter:=IMAP New transporter($server)

$boxList:=$transporter.getBoxList()

For each($box;$boxList)
  If($box.interesting)
    $split:=Split string($box.name;$transporter.getDelimiter())
    ALERT("新規メールが届いています: "+$split[$split.length-1])
  End if
End for each
```

.getDelimiter()

▶ 履歴

.getDelimiter() : Text

引数	タイプ		説明
戻り値	テキスト	<-	階層区切り文字

説明

.getDelimiter() 関数は、メールボックス名で階層レベルを区切るのに使用される文字を返します。

この区切り文字は以下のように使用することができます:

- 下層レベルのメールボックスを作成する
- メールボックスの階層内での上層・下層レベルを検索する

戻り値

メールボックス名の区切り文字

- 開いている接続がない場合、.getDelimiter() は接続を開きます。
- 接続が指定された時間 (IMAP New transporter 参照) 以上に使用されなかった場合には、.checkConnection() 関数が自動的に呼び出されます。

例題

```
var $transporter : 4D.IMAPTransporter
$transporter:=IMAP New transporter($server)

$boxList:=$transporter.getBoxList()

For each($box;$boxList)
  If($box.interesting)
    $split:=Split string($box.name;$transporter.getDelimiter())
    ALERT("新規メールが届いています: "+$split[$split.length-1])
  End if
End for each
```

.getMail()

▶ 履歴

.getMail(*msgNumber*: Integer { ; *options* : Object }) : Object
.getMail(*msgID*: Text { ; *options* : Object }) : Object

引数	タイプ		説明
<i>msgNumber</i>	整数	->	メッセージのシーケンス番号
<i>msgID</i>	テキスト	->	メッセージの固有ID
<i>options</i>	オブジェクト	->	メッセージ管理オプション
戻り値	オブジェクト	<-	Email オブジェクト

説明

.getMail() 関数は、IMAP_transporter が指定するメールボックス内の、*msgNumber* または *msgID* に対応するメールを Email オブジェクトとして返します。この関数を使用すると、メールのコンテンツをローカルで管理できるようになります。

最初の引数として、次のいずれかを渡すことができます:

- *msgNumber* に、取得するメッセージのシーケンス番号 (倍長整数) を渡します。
- *msgID* に、取得するメッセージの固有ID (テキスト) を渡します。

任意の *options* 引数として、メッセージの扱い方を定義する追加のオブジェクトを渡すことができます。次のプロパティを利用することができます:

プロパティ	タイプ	説明
<i>updateSeen</i>	boolean	true 時には、メールボックス内でメッセージを "既読" にします。false 時にはメッセージの状態は変化しません。デフォルト値: true
<i>withBody</i>	boolean	true を渡すとメッセージ本文を返します。false 時には、メッセージヘッダーのみが返されます。デフォルト値: true

- *msgID* 引数が存在しないメッセージを指定した場合、関数はエラーを生成し Null を返します。
- .selectBox() によって選択されたメールボックスがない場合、エラーが生成されます。
- 開いている接続がない場合、.getMail() は .selectBox() で最後に指定されたメールボックスへの接続を開きます。

戻り値

.getMail() は、以下の IMAP特有のプロパティを持つ Email オブジェクトを返します: *id*, *receivedAt*, および *size*.

例題

ID = 1のメッセージを取得します:

```

var $server : Object
var $info; $mail; $boxInfo : Variant
var $transporter : 4D.IMAPTransporter

$server:=New object
$server.host:="imap.gmail.com" // 必須
$server.port:=993
$server.user:="4d@gmail.com"
$server.password:="XXXXXXX"

// transporter を作成します
$transporter:=IMAP New transporter($server)

//メールボックスを選択します
$boxInfo:=$transporter.selectBox("Inbox")

// ID = 1 の Emailオブジェクトを取得します
$mail:=$transporter.getMail(1)

```

.getMails()

▶履歴

.getMails(*ids* : Collection { ; *options* : Object }) : Object
 .getMails(*startMsg* : Integer ; *endMsg* : Integer { ; *options* : Object }) : Object

引数	タイプ		説明
<i>ids</i>	Collection	->	メッセージID のコレクション
<i>startMsg</i>	Integer	->	先頭メッセージのシーケンス番号
<i>endMsg</i>	Integer	->	最後のメッセージのシーケンス番号
<i>options</i>	Object	->	メッセージ管理オプション
戻り値	Object	<-	次のコレクションを格納したオブジェクト: <ul style="list-style-type: none"> ● Email オブジェクト のコレクション ● 見つからなかったメッセージの ID または番号のコレクション

説明

.getMails() 関数は、 Email オブジェクトのコレクションを格納したオブジェクトを返します。

第一シンタックス:

.getMails(*ids* { ; *options* }) -> *result*

第一シンタックスを使用すると、メッセージID に基づいてメッセージを取得することができます。

ids 引数として、取得するメッセージID のコレクションを渡します。これらの ID は [.getMail\(\)](#) で取得することができます。

任意の *options* 引数を渡すと、返されるメッセージのパートを定義することができます。利用可能なプロパティについては、以下の オプション の表を参考ください。

第二シンタックス:

.getMails(*startMsg* ; *endMsg* { ; *options* }) -> *result*

第二シンタックスを使用すると、連続したレンジに基づいてメッセージを取得することができます。渡される値はメールボックス内のメッセージの位置を表します。

startMsg には、連続したレンジの最初のメッセージの番号に対応する 倍長整数 の値を渡します。負の値 (*startMsg* <= 0) を渡した場合、メールボックスの最初のメッセージが連続レンジの先頭メッセージとして扱われます。

endMsg には、連続レンジに含める最後のメッセージの番号に対応する 倍長整数 の値を渡します。負の値 (*startMsg* <= 0) を渡した場合、メー

ルボックスの最後のメッセージが連続レンジの最終メッセージとして扱われます。

任意の *options* 引数を渡すと、返されるメッセージのパートを定義することができます。

オプション

プロパティ	タイプ	説明
updateSeen	Boolean	true 時には、指定されたメッセージを "既読" にします。false 時にはメッセージの状態は変化しません。デフォルト値: true
withBody	Boolean	true を渡すと指定されたメッセージの本文を返します。false 時には、メッセージヘッダーのみが返されます。デフォルト値: true

- `.selectBox()` によって選択されたメールボックスがない場合、エラーが生成されます。
- 開いている接続がない場合、`.getMails()` は `.selectBox()` で最後に指定されたメールボックスへの接続を開きます。

戻り値

`.getMails()` は、以下のコレクションを格納したオブジェクトを返します。

プロパティ	タイプ	説明
list	Collection	<code>Email</code> オブジェクトのコレクション。Email オブジェクトが見つからない場合、空のコレクションが返されます。
notFound	Collection	使用したシンタックスによって返されるものが異なります: <ul style="list-style-type: none">第一シンタックス - 指定した ID のうち、存在しなかったメッセージの ID第二シンタックス - startMsg と endMsg の間の番号のうち、存在しなかったメッセージの番号 すべてのメッセージが見つかった場合には、空のコレクションが返されます。

例題

直近の 20件のメールを、"既読" ステータスを変更せずに取得します:

```
var $server,$boxInfo,$result : Object
var $transporter : 4D.IMAPTransporter

$server:=New object
$server.host:="imap.gmail.com" // 必須
$server.port:=993
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXX"

//create transporter
$transporter:=IMAP New transporter($server)

// メールボックスを選択します
$boxInfo:=$transporter.selectBox("INBOX")

If($boxInfo.mailCount>0)
    // 直近20件のメッセージのヘッダーを、"既読" にせずに取得します
    $result:=$transporter.getMails($boxInfo.mailCount-20;$boxInfo.mailCount;\ 
        New object("withBody";False;"updateSeen";False))
    For each($mail;$result.list)
        // ...
    End for each
End if
```

.getMIMEAsBlob()

▶ 履歴

.getMIMEAsBlob(*msgNumber* : Integer { ; *updateSeen* : Boolean }) : Blob
.getMIMEAsBlob(*msgID* : Text { ; *updateSeen* : Boolean }) : Blob

引数	タイプ		説明
<i>msgNumber</i>	Integer	->	メッセージのシーケンス番号
<i>msgID</i>	Text	->	メッセージの固有ID
<i>updateSeen</i>	Boolean	->	true 時には、メールボックス内でメッセージを "既読" にします。false 時にはメッセージの状態は変化しません。
戻り値	BLOB	<-	メールサーバーから返された MIME 文字列の BLOB

説明

`.getMIMEAsBlob()` 関数は、`IMAP_transporter` が指定するメールボックス内の、*msgNumber* または *msgID* に対応するメッセージの MIME コンテンツを格納した BLOB を返します。

最初の引数として、次のいずれかを渡すことができます：

- *msgNumber* に、取得するメッセージのシーケンス番号（倍長整数）を渡します。
- *msgID* に、取得するメッセージの固有ID（テキスト）を渡します。

任意の *updateSeen* 引数を渡すと、メールボックス内でメッセージが "既読" とマークされるかどうかを指定します。以下のものを渡すことができます：

- True - メッセージは "既読" とマークされます（このメッセージが読まれたことを表します）
- False - メッセージの "既読" ステータスは変化しません。

- *msgNumber* または *msgID* 引数が存在しないメッセージを指定した場合、関数は空の BLOB を返します。
- `.selectBox()` によって選択されたメールボックスがない場合、エラーが生成されます。
- 開いている接続がない場合、`.getMIMEAsBlob()` は `.selectBox()` で最後に指定されたメールボックスへの接続を開きます。

戻り値

`.getMIMEAsBlob()` は BLOB を返します。この BLOB はデータベースにアーカイブしたり、`MAIL Convert from MIME` コマンドを使用して `Email オブジェクト` へと変換したりすることができます。

例題

```
var $server : Object
var $boxInfo : Variant
var $blob : Blob
var $transporter : 4D.IMAPTransporter

$server:=New object
$server.host:="imap.gmail.com"
$server.port:=993
$server.user:="4d@gmail.com"
$server.password:="XXXXXXX"

// transporter を作成します
$transporter:=IMAP New transporter($server)

// メールボックスを選択します
$boxInfo:=$transporter.selectBox("Inbox")

// BLOB を取得します
$blob:=$transporter.getMIMEAsBlob(1)
```

.host

▶履歴
.host : Text

説明

.host プロパティは、ホストサーバーの名前または IPアドレスを格納します。この情報はメール通信 (SMTP、POP3、IMAP) に使用されます。

.logFile

▶履歴
.logFile : Text

説明

.logFile プロパティは、メール接続に対して定義された拡張ログファイル (あれば) へのフルパスを格納します。パスは、カレント Logs フォルダーを基準とした相対パス、あるいは絶対パスを指定できます。

SET DATABASE PARAMETER コマンドで有効化される通常のログファイルとは異なり、拡張ログファイルはすべての送信されたメールの MIMEコンテンツを保存し、サイズ制限がありません。拡張ログファイルの詳細については、以下の章をそれぞれ参照ください:

- SMTP 接続 - [4DSMTPLog.txt](#)
- POP3 接続 - [4DPOP3Log.txt](#)
- IMAP 接続 - [4DIMAPLog.txt](#)

.move()

▶履歴
.move(msgsIDs : Collection ; destinationBox : Text) : Object
.move(allMsgs : Integer ; destinationBox : Text) : Object

引数	タイプ		説明
msgsIDs	Collection	->	メッセージの固有ID のコレクション (テキスト)
allMsgs	Integer	->	IMAP all : 選択されたメールボックスの全メッセージ
destinationBox	Text	->	メッセージの移動先のメールボックス
戻り値	Object	<-	move処理のステータス

説明

.move() 関数は、msgsIDs または allMsgs で定義されたメッセージを IMAP サーバーの destinationBox へと移動します。

以下のものを渡すことができます:

- msgsIDs には、移動するメッセージの固有ID を格納したコレクション
- allMsgs には、選択されているメールボックスの全メッセージを移動するための定数 (倍長整数型):

destinationBox には、メッセージの移動先メールボックスの名称をテキスト値で渡すことができます。

RFC [8474](#) に準拠している IMAPサーバーでのみ、この関数はサポートされます。

返されるオブジェクト

この関数は、IMAP ステータスを表すオブジェクトを返します:

プロパティ		タイプ	説明
success		Boolean	処理が正常に終わった場合には true、それ以外は false
statusText		Text	IMAPサーバーから返されたステータスマッセージ、または 4Dエラースタック内に返された最後のエラー
errors		Collection	4Dエラースタック (IMAPサーバーレスポンスが受信できた場合には返されません)
	[].errcode	Number	4Dエラーコード
	[].message	Text	4Dエラーの詳細
	[].componentSignature	Text	エラーを返した内部コンポーネントの署名

例題 1

選択されたメッセージを移動します:

```

var $server;$boxInfo;$status : Object
var $mailIds : Collection
var $transporter : 4D.IMAPTransporter

$server:=New object
$server.host:="imap.gmail.com" // 必須
$server.port:=993
$server.user:="4d@gmail.com"
$server.password:="XXXXXXX"

$transporter:=IMAP New transporter($server)

// メールボックスを選択します
$boxInfo:=$transporter.selectBox("inbox")

// メッセージの固有IDのコレクションを取得します
$mailIds:=$transporter.searchMails("subject \\"4D new feature:\\"")

// カレントメールボックス内で見つかったメッセージを "documents" メールボックスに移動します
$status:=$transporter.move($mailIds;"documents")

```

例題 2

カレントメールボックスの全メッセージを移動します:

```

var $server;$boxInfo;$status : Object
var $transporter : 4D.IMAPTransporter

$server:=New object
$server.host:="imap.gmail.com" // 必須
$server.port:=993
$server.user:="4d@gmail.com"
$server.password:="XXXXXXX"

$transporter:=IMAP New transporter($server)

// メールボックスを選択します
$boxInfo:=$transporter.selectBox("inbox")

// カレントメールボックスの全メッセージを "documents" メールボックスに移動します
$status:=$transporter.move(IMAP all;"documents")

```

.numToID()

▶履歴

.numToID(*startMsg* : Integer ; *endMsg* : Integer) : Collection

引数	タイプ		説明
<i>startMsg</i>	Integer	->	先頭メッセージのシーケンス番号
<i>endMsg</i>	Integer	->	最後のメッセージのシーケンス番号
戻り値	Collection	<-	固有ID のコレクション

説明

.numToID() 関数は、現在選択されているメールボックスにおいて、 *startMsg* および *endMsg* で指定された連続した範囲のメッセージのシーケンス番号を IMAP 固有IDへと変換します。

startMsg には、連続したレンジの最初のメッセージの番号に対応する 倍長整数 の値を渡します。負の値 (*startMsg* <= 0) を渡した場合、メールボックスの最初のメッセージが連続レンジの先頭メッセージとして扱われます。

endMsg には、連続レンジに含める最後のメッセージの番号に対応する 倍長整数 の値を渡します。負の値 (*startMsg* <= 0) を渡した場合、メールボックスの最後のメッセージが連続レンジの最終メッセージとして扱われます。

戻り値

メソッドは文字列 (固有ID) のコレクションを返します。

例題

```
var $transporter : 4D.IMAPTransporter
var $server;$boxInfo;$status : Object
var $mailIds : Collection

$server:=New object
$server.host:="imap.gmail.com" // 必須
$server.port:=993
$server.user:="4d@gmail.com"
$server.password:="XXXXXXX"

$transporter:=IMAP New transporter($server)

// メールボックスを選択します
$boxInfo:=$transporter.selectBox("inbox")

// 最も古い 5通のメッセージID を取得します
$mailIds:=$transporter.numToID(({$boxInfo.mailCount}-5);$boxInfo.mailCount)

// カレントメールボックスからメッセージを削除します
$status:=$transporter.delete($mailIds)
```

.removeFlags()

▶履歴

.removeFlags(*msgIDs* : Collection ; *keywords* : Object) : Object

.removeFlags(*msgIDs* : Text ; *keywords* : Object) : Object

.removeFlags(*msgIDs* : Longint ; *keywords* : Object) : Object

引数	タイプ		説明
msgIDs	Collection	->	文字列のコレクション: メッセージの固有ID (テキスト型) テキスト: メッセージの固有ID 倍長整数 (IMAP all): 選択されたメールボックス内の全メッセージ
keywords	Object	->	削除するキーワードフラグ
戻り値	Object	<-	removeFlags処理のステータス

説明

.removeFlags() 関数は、 msgIDs のメッセージに対して、 keywords で指定したフラグを削除します。

msgIDs には、以下のいずれかを渡すことができます:

- 指定するメッセージの固有ID を格納した コレクション
- 単一のメッセージの固有ID (テキスト)
- 以下の定数 (longint) を使用することで、選択されているメールボックスの全メッセージを指定することができます:

定数	値	説明
IMAP all	1	選択されたメールボックスの全メッセージを選択します

keywords には、 msgIDs 引数で指定したメッセージから削除するフラグのキーワード値を格納したオブジェクトを渡します。次のキーワードを渡すことができます:

引数	タイプ	説明
\$draft	Boolean	メッセージの "draft" フラグを削除するには true
\$seen	Boolean	メッセージの "seen" フラグを削除するには true
\$flagged	Boolean	メッセージの "flagged" フラグを削除するには true
\$answered	Boolean	メッセージの "answered" フラグを削除するには true
\$deleted	Boolean	メッセージの "deleted" フラグを削除するには true

false値は無視されます。

返されるオブジェクト

この関数は、IMAP ステータスを表すオブジェクトを返します:

プロパティ		タイプ	説明
success		Boolean	処理が正常に終わった場合には true、それ以外は false
statusText		Text	IMAPサーバーから返されたステータスマッセージ、または 4Dエラースタック内に返された最後のエラー
errors		Collection	4Dエラースタック (IMAPサーバーレスポンスが受信できた場合には返されません)
	[].errcode	Number	4Dエラーコード
	[].message	Text	4Dエラーの詳細
	[].componentSignature	Text	エラーを返した内部コンポーネントの署名

例題

```

var $options;$transporter;$boxInfo;$status : Object

$options:=New object
$options.host:="imap.gmail.com"
$options.port:=993
$options.user:="4d@gmail.com"
$options.password:="xxxxxx"

// transporter を作成します
$transporter:=IMAP New transporter($options)

// メールボックスを選択します
$boxInfo:=$transporter.selectBox("INBOX")

// INBOX の全メッセージを未読にします
$flags:=New object
$flags["$seen"]:=True
$status:=$transporter.removeFlags(IMAP all;$flags)

```

.renameBox()

▶履歴

.renameBox(*currentName* : Text ; *newName* : Text) : Object

引数	タイプ		説明
<i>currentName</i>	Text	->	カレントメールボックスの名称
<i>newName</i>	Text	->	新しいメールボックス名
戻り値	Object	<-	renameBox処理のステータス

説明

.renameBox() 関数は、IMAPサーバー上でメールボックスの名称を変更します。存在しないメールボックスの名称を変更しようとしたり、すでに使われているメールボックス名に変更しようとしたりすると、エラーが生成されます。

currentName には、名称変更するメールボックスの名前を渡します。

メールボックスの新しい名称は *newName* に渡します。

返されるオブジェクト

この関数は、IMAP ステータスを表すオブジェクトを返します:

プロパティ		タイプ	説明
<i>success</i>		Boolean	処理が正常に終わった場合には true、それ以外は false
<i>statusText</i>		Text	IMAPサーバーから返されたステータスマッセージ、または 4Dエラースタック内に返された最後のエラー
<i>errors</i>		Collection	4Dエラースタック (IMAPサーバーレスポンスが受信できた場合には返されません)
	[].errcode	Number	4Dエラーコード
	[].message	Text	4Dエラーの詳細
	[].componentSignature	Text	エラーを返した内部コンポーネントの署名

例題

"Invoices" メールボックスを "Bills" に名称変更します:

```

var $pw : text
var $options; $transporter; $status : object

$options:=New object

$pw:=Request("パスワードを入力してください:")

If(OK=1) $options.host:="imap.gmail.com"
$options.user:="test@gmail.com"
$options.password:=$pw

$transporter:=IMAP New transporter($options)

// メールボックスの名称変更
$status:=$transporter.renameBox("Invoices"; "Bills")

If ($status.success)
    ALERT("メールボックスの名称が変更されました。")
Else
    ALERT("エラー: "+$status.statusText)
End if
End if

```

.port

▶履歴

.port : Integer

説明

.port プロパティは、メール通信に使用されるポート番号を格納します。 SMTP New transporter や POP3 New transporter 、 IMAP New transporter のコマンドで transporter オブジェクトを作成する際に使用される server オブジェクトにおいて、このプロパティが指定されなかった場合に使用されるポートは次のとおりです：

- SMTP - 587
- POP3 - 995
- IMAP - 993

.searchMails()

▶履歴

.searchMails(searchCriteria : Text) : Collection

引数	タイプ		説明
searchCriteria	Text	->	検索条件
戻り値	Collection	<-	メッセージ番号のコレクション

説明

この関数は、IMAP プロトコル の仕様に基づいています。

.searchMails() 関数は、カレントメールボックスにおいて searchCriteria の検索条件に合致するメッセージを検索します。searchCriteria 引数には、一つ以上の検索キーを格納します。

searchCriteria はテキスト型の引数で、一つ以上の検索キー（詳細は後述の [利用可能な検索キー](#) 参照）を格納し、検索する値を渡します（渡さない場合もあります）。検索キーは单一または複数の項目からなります。たとえば：

```
SearchKey1 = FLAGGED  
SearchKey2 = NOT FLAGGED  
SearchKey3 = FLAGGED DRAFT
```

文字の大小は通常区別されません。

- `searchCriteria` 引数が null 文字列の場合、検索は "すべてを選択" と同等です。
- 引数が複数の検索キーを格納している場合、それらすべてに合致する和集合 (AND) が検索結果になります。

```
searchCriteria = FLAGGED FROM "SMITH"
```

... この検索結果は ¥Flagged フラグが設定されていて、かつ Smith から送られたメッセージをすべて返します。

- OR および NOT 演算子を、以下のように使用することができます:

```
searchCriteria = OR SEEN FLAGGED
```

... ¥Seen フラグが設定されている、あるいは ¥Flagged フラグが設定されているメッセージをすべて返します。

```
searchCriteria = NOT SEEN
```

... ¥Seen フラグが設定されていないメッセージをすべて返します。

```
searchCriteria = HEADER CONTENT-TYPE "MIXED" NOT HEADER CONTENT-TYPE "TEXT" ...
```

... content-type ヘッダーが "Mixed" を格納しているもののうち、"Text" は格納していないメッセージを返します。

```
searchCriteria = HEADER CONTENT-TYPE "E" NOT SUBJECT "o" NOT HEADER CONTENT-TYPE "MIXED"
```

... content-type ヘッダーが "e" を格納しているもののうち、Subject ヘッダーが "o" を格納していないもの、かつ content-type ヘッダーが "Mixed" でないメッセージを返します。

最後の 2例については、最初の検索キーリストのカッコを取り除いてしまうと検索結果が異なることに注意してください。

- `searchCriteria` 引数には任意の [CHARSET] 指定を含めることができます。これは "CHARSET" という単語の後に実際の文字コード [CHARSET] (US ASCII, ISO-8859 など) が続きます。これは `searchCriteria` 文字列の文字コードを指定します。そのため、[CHARSET] 指定を使用する場合には `searchCriteria` 文字列を指定された文字コードへと変換する必要があります (詳細については `CONVERT FROM TEXT` または `Convert to text` コマンドを参照ください)。デフォルトでは、`searchCriteria` 引数に拡張された文字列が含まれていた場合には 4D はそれを Quotable Printable へとエンコードします。

```
searchCriteria = CHARSET "ISO-8859" BODY "Help"
```

... これは、検索条件に iso-8859 文字コードを使用し、必要に応じてサーバーは検索前に検索条件をこの文字コードに変換しなければならない、ということを意味します。

検索する値の型について

検索キーによっては、次の型の検索値が必要となる場合があります:

- 日付値の検索キー: `date` は日付を指定する文字列で、以下のようにフォーマットされている必要があります: `date-day+-+date-month+-+date-year`。ここで `date-day` は日付の数値 (最大2桁) を意味し、`date-month` は月の名前 (Jan/Feb/Mar/Apr/May/Jun/Jul/Aug/Sep/Oct/Dec) を意味し、`date-year` は年 (4桁) を意味します。例: `searchCriteria = SENTBEFORE 1-Feb-2000` (日付は特殊文字を含まないため、通常は引用符でくる必要はありません)

- 文字列値の検索キー: string はあらゆる文字列を含みうるため、引用符でくらなければなりません。文字列が特殊文字（スペース文字など）をまったく含まない場合には、引用符で括る必要はありません。このような文字列を引用符でくくることは、渡した文字列値が正確に解釈されることを保証します。例: `searchCriteria = FROM "SMITH"`

文字列を使用するすべての検索キーに対し、フィールドの文字列に検索キーが含まれる場合には検索に合致したとみなされます。合致は文字の大小を区別しません。

- field-name 値の検索キー: field-name はヘッダーフィールドの名称です。例: `searchCriteria = HEADER CONTENT-TYPE "MIXED"`
- フラグ値の検索キー: flag は一つ以上のキーワードを（標準のフラグを含めて）受け入れます。複数指定する場合にはスペースで区切れます。例: `searchCriteria = KEYWORD \Flagged \Draft`
- メッセージセット値の検索キー: 複数のメッセージを識別します。メッセージシーケンス番号は、1 から始まりメールボックスのメッセージの総数までの連続した番号です。個別の番号はカンマで区切れます。コロンは、その前後の番号を含めた連続した番号を指定します。例:
`2,4:7,9,12:*` は、15通あるメールボックスの場合に `2,4,5,6,7,9,12,13,14,15` を指定します。`searchCriteria = 1:5 ANSWERED` は、メッセージシーケンス番号 1 から 5番のメッセージのうち、¥Answered フラグが設定されているメッセージを検索します。
`searchCriteria= 2,4 ANSWERED` は、メッセージセレクション（メッセージ番号 2番と4番）のうち、¥Answered フラグが設定されているメッセージを検索します。

利用可能な検索キー

ALL: メールボックスの全メッセージ
ANSWERED: ¥Answered フラグが設定されたメッセージ
UNANSWERED: ¥Answered フラグが設定されていないメッセージ
DELETED: ¥Deleted フラグが設定されたメッセージ
UNDELETED: ¥Deleted フラグが設定されていないメッセージ
DRAFT: ¥Draft フラグが設定されているメッセージ
UNDRAFT: ¥Draft フラグが設定されていないメッセージ
FLAGGED: ¥Flagged フラグが設定されているメッセージ
UNFLAGGED: ¥Flagged フラグが設定されていないメッセージ
RECENT: ¥Recent フラグが設定されているメッセージ
OLD: ¥Recent フラグが設定されていないメッセージ
SEEN: ¥Seen フラグが設定されているメッセージ
UNSEEN: ¥Seen フラグが設定されていないメッセージ
NEW: ¥Recent フラグが設定されているが ¥Seen フラグが設定されていないメッセージ。これは機能的には “(RECENT UNSEEN)” と同じです。
KEYWORD ***flag**: 指定されたキーワードが設定されているメッセージ
UNKEYWORD ***flag**: 指定されたキーワードが設定されていないメッセージ
BEFORE ***date**: 内部の日付が指定日より前のメッセージ
ON ***date**: 内部の日付が指定日に合致するメッセージ
SINCE ***date**: 内部の日付が指定日より後のメッセージ
SENTBEFORE ***date**: 日付ヘッダーが指定日より前のメッセージ
SENTON ***date**: 日付ヘッダーが指定日に合致するメッセージ
SENTSINCE ***date**: 日付ヘッダーが指定日以降のメッセージ
TO ***string**: TO ヘッダーに指定文字列が含まれているメッセージ
FROM ***string**: FROM ヘッダーに指定文字列が含まれているメッセージ
CC ***string**: CC ヘッダーに指定文字列が含まれているメッセージ
BCC ***string**: BCC ヘッダーに指定文字列が含まれているメッセージ
SUBJECT ***string**: 件名ヘッダーに指定文字列が含まれているメッセージ
BODY ***string**: メッセージ本文に指定文字列が含まれているメッセージ
TEXT ***string**: ヘッダーまたはメッセージ本文に指定文字列が含まれているメッセージ
HEADER field-name ***string**: 指定フィールド名のヘッダーを持ち、そのフィールド内に指定文字列が含まれているメッセージ
UID ***message-UID**: 指定された固有識別子に対応する固有識別子を持つメッセージ
LARGER ***n**: 指定バイト数以上のサイズを持つメッセージ
SMALLER ***n**: 指定バイト数以下のサイズを持つメッセージ
NOT ***search-key**: 指定検索キーに合致しないメッセージ
OR search-key1 ***search-key2**: いづれかの検索キーに合致するメッセージ

.selectBox()

▶履歴

`.selectBox(name : Text { ; state : Integer }) : Object`

引数	タイプ		説明
name	テキスト	->	メールボックスの名称
state	整数	->	メールボックスのアクセス状態
戻り値	オブジェクト	<-	boxInfo オブジェクト

説明

.selectBox() 関数は、name に指定したメールボックスをカレントメールボックスとして選択します。この関数を使用するとメールボックスに関する情報を取得することができます。

カレントメールボックスを変更せずに、メールボックスから情報を取得するには、.getBoxInfo() を使用します。

name には、アクセスするメールボックスの名前を渡します。この名称は明確な左から右への階層を表し、特定の区切り文字でレベルを区分けします。この区切り文字は .getDelimiter() 関数で調べることができます。

任意の state 引数を渡すと、メールボックスへのアクセスタイプを定義できます。取りうる値は以下の通りです:

定数	値	説明
IMAP read only state	1	選択されたメールボックスは読み取り専用権限でアクセスされます。新しいメッセージを表す "新着" フラグはそのまま変化しません。
IMAP read write state	0	選択されたメールボックスは読み書き可能権限でアクセスされます。メッセージは "既読" と判断され、"新着" フラグは失われます。(デフォルト値)

- name 引数が存在しないメールボックスを指定した場合、関数はエラーを生成し Null を返します。
- 開いている接続がない場合、.selectBox() は接続を開きます。
- 接続が指定された時間 (IMAP New transporter 参照) 以上に使用されなかった場合には、.checkConnection() 関数が自動的に呼び出されます。

返されるオブジェクト

返される boxInfo オブジェクトには、以下のプロパティが格納されています:

プロパティ	タイプ	説明
name	テキスト	メールボックスの名称
mailCount	number	メールボックス内のメッセージの数
mailRecent	number	"recent" フラグがついたメッセージの数

例題

```
var $server; $boxinfo : Object
$server:=New object
$server.host:="imap.gmail.com" // 必須
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXX"

var $transporter : 4D.IMAPTransporter
$transporter:=IMAP New transporter($server)
$boxInfo:=$transporter.selectBox("INBOX")
```

.subscribe()

▶ 履歴

.subscribe(name : Text) : Object

引数	タイプ		説明
name	テキスト	->	メールボックスの名称
戻り値	オブジェクト	<-	subscribe処理のステータス

説明

.subscribe() 関数は、IMAPサーバーの購読メールボックスとして任意のメールボックスを追加・削除します。利用可能なメールボックスが大量にある場合、すべてを取得するのを避けるため、確認したいメールボックスだけを購読することができます。

name には、購読するメールボックスの名前を渡します。

返されるオブジェクト

この関数は、IMAP ステータスを表すオブジェクトを返します:

プロパティ		タイプ	説明
success		Boolean	処理が正常に終わった場合には true、それ以外は false
statusText		Text	IMAPサーバーから返されたステータスマッセージ、または 4Dエラースタック内に返された最後のエラー
errors		Collection	4Dエラースタック (IMAPサーバーレスポンスが受信できた場合には返されません)
	[].errcode	Number	4Dエラーコード
	[].message	Text	4Dエラーの詳細
	[].componentSignature	Text	エラーを返した内部コンポーネントの署名

例題

"Bills" 階層下の "Atlas Corp" メールボックスを購読します:

```
var $pw; $name : text
var $options; $transporter; $status : object

$options:=New object

$pw:=Request("パスワードを入力してください:")

If(OK=1) $options.host:="imap.gmail.com"
$options.user:="test@gmail.com"
$options.password:=$pw

$transporter:=IMAP New transporter($options)

$name:="Bills"+$transporter.getDelimiter()+"Atlas Corp"
$status:=$transporter.subscribe($name)

If ($status.success)
  ALERT("メールボックスの購読に成功しました。")
Else
  ALERT("エラー: "+$status.statusText)
End if
End if
```

.unsubscribe()

▶ 覆歴

.unsubscribe(name : Text) : Object

引数	タイプ		説明
name	Text	->	メールボックスの名称
戻り値	Object	<-	unsubscribe処理のステータス

説明

.unsubscribe() 関数は、指定したメールボックスを購読メールボックスから削除します。これにより、確認するメールボックスの数を減らせます。

name には、購読を解除するメールボックスの名前を渡します。

返されるオブジェクト

この関数は、IMAP ステータスを表すオブジェクトを返します:

プロパティ		タイプ	説明
success		Boolean	処理が正常に終わった場合には true、それ以外は false
statusText		Text	IMAPサーバーから返されたステータスマッセージ、または 4Dエラースタック内に返された最後のエラー
errors		Collection	4Dエラースタック (IMAPサーバーレスポンスが受信できた場合には返されません)
	[].errcode	Number	4Dエラーコード
	[].message	Text	4Dエラーの詳細
	[].componentSignature	Text	エラーを返した内部コンポーネントの署名

例題

"Bills" 階層下の "Atlas Corp" メールボックスの購読を解除します:

```
var $pw; $name : text
var $options; $transporter; $status : object

$options:=New object

$pw:=Request("パスワードを入力してください:")

If(OK=1) $options.host:="imap.gmail.com"
$options.user:="test@gmail.com"
$options.password:=$pw

$transporter:=IMAP New transporter($options)

$name:="Bills"+$transporter.getDelimiter()+"Atlas Corp"
$status:=$transporter.unsubscribe($name)

If ($status.success)
  ALERT("メールボックスの購読を解除しました。")
Else
  ALERT("エラー: "+$status.statusText)
End if
End if
```

.user

▶ 補足

.user : Text

説明

.user プロパティは、メールサーバーでの認証に使用されたユーザー名を格納します。

MailAttachment

Attachment オブジェクトによって、`Email` オブジェクト内のファイルを参照することができます。MailAttachment オブジェクトは `MAIL New attachment` コマンドによって作成されます。

Attachment オブジェクト

Attachment オブジェクトは、次の読み取り専用プロパティや、関数を提供します：

<code>.cid : Text</code>	添付ファイルの ID
<code>.disposition : Text</code>	<code>Content-Disposition</code> ヘッダーの値
<code>.getContent() : 4D.Blob</code>	添付オブジェクトの中身を <code>4D.Blob</code> オブジェクトとして返します
<code>.name : Text</code>	添付ファイルの名前と拡張子
<code>.path : Text</code>	添付ファイルの POSIX パス
<code>.platformPath : Text</code>	カレントプラットフォームのシンタックスで表現されたファイルのパス
<code>.type : Text</code>	添付ファイルの <code>content-type</code>

MAIL New attachment

▶ 補足

```
MAIL New attachment( file : 4D.File { ; name : Text {; cid : Text{ ; type : Text { ; disposition :Text } } } } ) :  
4D.MailAttachment  
MAIL New attachment( zipFile : 4D.ZipFile { ; name : Text {; cid : Text{ ; type : Text { ; disposition :Text } } } } ) :  
4D.MailAttachment  
MAIL New attachment( blob : 4D.Blob { ; name : Text {; cid : Text{ ; type : Text { ; disposition :Text } } } } ) :  
4D.MailAttachment  
MAIL New attachment( path : Text { ; name : Text {; cid : Text{ ; type : Text { ; disposition :Text } } } } ) :  
4D.MailAttachment
```

引数	タイプ		説明
file	4D.File	->	添付ファイル
zipFile	4D.ZipFile	->	添付 Zipファイル
blob	4D.Blob	->	添付を格納した BLOB
path	Text	->	添付ファイルのパス
name	Text	->	メールクライアントが添付を指定するのに使用する名前 + 拡張子
cid	Text	->	添付の ID (HTMLメッセージのみ)、あるいは cid が不要な場合は "" (空の文字列)
type	Text	->	content-type ヘッダーの値
disposition	Text	->	content-disposition ヘッダーの値: "inline" あるいは "attachment"
戻り値	4D.MailAttachment	<-	Attachment オブジェクト

説明

`MAIL New attachment` コマンドは、[Email オブジェクト](#) に追加することができる添付オブジェクトを作成します。

添付を定義するには、次のパラメーターが使えます:

- `file`: 添付ファイルを格納する `4D.File` オブジェクトを渡します。
- `zipfile`: 添付ファイルを格納する `4D.ZipFile` オブジェクトを渡します。
- `blob`: 添付そのものを `4D.Blob` に格納して渡します。
- `path`: システムシンタックスで表現された添付ファイルのパスを テキスト 値で渡します。完全なパス名、または単純なファイル名を渡すことができます (ファイル名のみの場合、4D はプロジェクトファイルと同じディレクトリ内を検索します)。

任意の `name` 引数として、添付を指定するためにメールクライアントが使用する名前と拡張子を渡すことができます。`name` が省略された場合:

- ファイルパスを渡していれば、そのファイル名と拡張子が使用されます。
- BLOB を渡していれば、拡張子がないランダムな名前が自動的に生成されます。

任意の `cid` 引数を使用すると、添付ファイルの内部ID を渡すことができます。この ID は `Content-Id` ヘッダーの値で、HTMLメッセージにおいてのみ使用されます。cid を使い、`` のような HTMLタグによってメッセージ本文で定義された参照と添付ファイルが紐づけられます。これはつまり、添付ファイルの中身 (例: ピクチャー) がメールクライアント上ではメッセージ本文内に表示されるべきであることを意味しています。最終的な表示は、メールクライアントによって若干異なる可能性があります。cid を使用したくない場合、空の文字列を引数として渡します。

任意の `type` 引数を渡すと、添付ファイルの `content-type` を明示的に設定することができます。たとえば、MIMEタイプを定義する文字列 ("video/mpeg"など) を渡すことができます。この `content-type` の値は拡張子とは関係なく添付ファイルに対して設定されます。MIMEタイプについての詳細は、[Wikipedia 上の MIME に関するページ](#) を参照ください。

この引数が省略された場合、あるいはこの引数に空の文字列が渡された場合はデフォルトで、添付ファイルの `content-type` は拡張子に基づいて設定されます。主な MIMEタイプについては、以下のルールが適用されます:

拡張子	Content-Type
jpg, jpeg	image/jpeg
png	image/png
gif	image/gif
pdf	application/pdf
doc	application/msword
xls	application/vnd.ms-excel
ppt	application/vnd.ms-powerpoint
zip	application/zip
gz	application/gzip
json	application/json
js	application/javascript
ps	application/postscript
xml	application/xml
htm, html	text/html
mp3	audio/mpeg
その他	application/octet-stream

任意の *disposition* 引数を渡して、添付ファイルの `content-disposition` ヘッダーを指定できます。"Mail" 定数テーマ内の、以下の定数のいずれか 1つを渡すことができます：

定数	値	説明
mail disposition attachment	"attachment"	Content-disposition ヘッダーの値を "attachment" に設定します。これは添付ファイルはメッセージ内でリンクとして提供される必要があることを意味します。
mail disposition inline	"inline"	Content-disposition ヘッダーの値を "inline" に設定します。これは添付ファイルはメッセージ本文内の、"cid" の位置にレンダリングされる必要があることを意味します。レンダリングの結果はメールクライアントによって異なります。

disposition 引数が省略された場合はデフォルトで：

- *cid* 引数が使われていた場合、`Content-disposition` ヘッダーは "inline" に設定されます。
- *cid* 引数が渡されていない、あるいは空の文字列が渡されていた場合、`Content-disposition` ヘッダーは "attachment" に設定されます。

例題 1

ユーザーが選択したファイルを添付し、HTML 本文に画像を埋め込んだメールを送信します：

```

$doc:=Select document("");%;"添付するファイルを選択してください";0)
If (OK=1) // もしドキュメントが選択されていれば

C_OBJECT($email;$server;$transporter)

$server:=New object
$server.host:="smtp.mail.com"
$server.user:="test_user@mail.com"
$server.password:="p@ssw@rd"
$transporter:=SMTP New transporter($server)

$email:=New object
$email.from:="test_user@mail.com"
$email.to:="test_user@mail.com"
$email.subject:="添付の付いたテストメッセージです"

// ファイルをダウンロードするリンクを追加します
$email.attachments:=New collection(MAIL New attachment(Document))
// インラインピクチャーを挿入します (cid を使用)
$email.attachments[1]:=MAIL New attachment("c:\\\\Pictures\\\\4D.jpg";"";"4D")

$email.htmlBody:="<html>" + \
"<body>Hello World!" + \
"<img src='cid:4D' >" + \
"</body>" + \
"</head>" + \
"</html>"

$transporter.send($email) // メールを送信します

End if

```

例題 2

4D Write Pro エリアを添付したメールを送信します:

```

C_BLOB($blob)
WP EXPORT VARIABLE(WPArea;$blob;wk docx)

C_OBJECT($email;$server;$transporter)

$server:=New object
$server.host:="smtp.mail.com"
$server.user:="user@mail.com"
$server.password:="p@ssw@rd"
$transporter:=SMTP New transporter($server)

$email:=New object
$email.from:="user@mail.com"
$email.to:="customer@mail.com"
$email.subject:="新規年次レポート"
$email.textBody:="添付のとおり、新しい年次レポートをご連絡します。"
$email.attachments:=New collection(MAIL New attachment($blob;"Annual report.docx"))

$transporter.send($email)

```

4D.MailAttachment.new()

▶ 履歴

4D.MailAttachment.new(file : 4D.File { ; name : Text {; cid : Text{ ; type : Text { ; disposition :Text } } } }) :

```

4D.MailAttachment
4D.MailAttachment.new( zipFile : 4D.ZipFile { ; name : Text {; cid : Text{ ; type : Text { ; disposition :Text } } } } ) :
4D.MailAttachment
4D.MailAttachment.new( blob : 4D.Blob { ; name : Text {; cid : Text{ ; type : Text { ; disposition :Text } } } } ) :
4D.MailAttachment
4D.MailAttachment.new( path : Text { ; name : Text {; cid : Text{ ; type : Text { ; disposition :Text } } } } ) :
4D.MailAttachment

```

引数	タイプ		説明
file	4D.File	->	添付ファイル
zipFile	4D.ZipFile	->	添付 Zipファイル
blob	4D.Blob	->	添付を格納した BLOB
path	Text	->	添付ファイルのパス
name	Text	->	メールクライアントが添付を指定するのに使用する名前 + 拡張子
cid	Text	->	添付の ID (HTMLメッセージのみ)、あるいは cid が不要な場合は "" (空の文字列)
type	Text	->	content-type ヘッダーの値
disposition	Text	->	content-disposition ヘッダーの値: "inline" あるいは "attachment"
戻り値	4D.MailAttachment	<-	Attachment オブジェクト

説明

`4D.MailAttachment.new()` 関数は、`4D.MailAttachment` 型の新規オブジェクト作成して返します。この関数の機能は、[MAIL New attachment](#) コマンドと同一です。

.cid

`.cid` : Text

説明

`.cid` プロパティは、添付ファイルの IDを格納します。このプロパティは HTMLメッセージでのみ使用されます。このプロパティがない場合、ファイルは単なる添付 (リンク) として管理されます。

.disposition

`.disposition` : Text

説明

`.disposition` プロパティは、`Content-Disposition` ヘッダーの値を格納します。二つの値が利用可能です:

- "inline": 添付ファイルはメッセージコンテンツ内に、"cid"の場所にレンダリングされます。レンダリングの結果はメールクライアントによって異なります。
- "attachment": 添付ファイルはメッセージ内でリンクとして提供されます。

.getContent()

`.getContent()` : 4D.Blob

引数	タイプ		説明
戻り値	4D.Blob	<-	添付の中身

説明

`.getContent()` 関数は、添付オブジェクトの中身を `4D.Blob` オブジェクトとして返します。 [MAIL Convert from MIME](#) コマンドによって取得した添付オブジェクトに対して、この関数を使用することができます。

.name

`.name` : Text

説明

`.name` プロパティは、添付ファイルの名前と拡張子を格納します。 [MAIL New attachment](#) コマンドで別の名称を指定しなかった場合のデフォルトは、ファイルの名称です。

.path

`.path` : Text

説明

`.path` プロパティは、添付ファイルの POSIXパス(存在すれば) を格納します。

.platformPath

▶ [履歴](#)

`.platformPath` : Text

説明

`.platformPath` プロパティは、カレントプラットフォームのシンタックスで表現されたファイルのパスを返します。

.type

`.type` : Text

説明

`.type` プロパティは、添付ファイルの `content-type` を格納します。 [MAIL New attachment](#) コマンドにて、このタイプが明示的に渡されていない場合、`content-type` はファイルの拡張子に基づきます。

POP3Transporter

`POP3Transporter` クラスを使って、POP3 メールサーバーからメッセージを取得することができます。

POP3 Transporter オブジェクト

POP3 Transporter オブジェクトは [POP3 New transporter](#) コマンドによってインスタンス化されます。これらは、次のプロパティや関数を持ちます：

`.acceptUnsecureConnection : Boolean`

暗号化されていない接続の確立が許可されれば `true`

`.authenticationMode : Text`

メールサーバーのセッションを開くのに使用される認証モード

`.checkConnection() : Object`

transporter オブジェクトが保存する情報を使用して接続をチェックします

`.connectionTimeOut : Integer`

サーバー接続の確立までに待機する最長時間 (秒単位)

`.delete(msgNumber : Integer)`

`msgNumber` で指定したメールメッセージに対して、POP3サーバーから削除するためのフラグを立てます

`.getBoxInfo() : Object`

対象の `POP3 transporter` が指定するメールボックスに対応する `boxInfo` オブジェクトを返します

`.getEmail(msgNumber : Integer) : Object`

`POP3 transporter` が指定するメールボックス内の、`msgNumber` に対応するメールを `Email` オブジェクトとして返します

`.getEmailInfo(msgNumber : Integer) : Object`

`POP3 transporter` が指定するメールボックス内の、`msgNumber` に対応するメールの `mailInfo` オブジェクトを返します

`.getMailInfoList() : Collection`

`POP3 transporter` が指定するメールボックス内の全メッセージについて記述した `mailInfo` オブジェクトのコレクションを返します

`.getMIMEAsBlob(msgNumber : Integer) : Blob`

`POP3_transporter` が指定するメールボックス内の、`msgNumber` に対応するメッセージの MIMEコンテンツを格納した BLOB を返します

`.host : Text`

ホストサーバーの名前または IPアドレス

`.logFile : Text`

メール接続に対して定義された拡張ログファイル (あれば) へのフルパス

`.port : Integer`

メール通信に使用されるポート番号

`.undeleteAll()`

`POP3_transporter` 内のメールに設定された削除フラグをすべて除去します

`.user : Text`

メールサーバーでの認証に使用されたユーザー名

POP3 New transporter

▶履歴

POP3 New transporter(*server* : Object) : 4D.POP3Transporter

引数	タイプ		説明
<i>server</i>	object	->	メールサーバー情報
戻り値	4D.POP3Transporter	<-	POP3 transporter オブジェクト

説明

.getMail() 関数は、POP3 transporter が指定するメールボックス内の、msgNumber に対応するメールを Email オブジェクトとして返します。返された transporter オブジェクトは、通常メールの受信に使用されます。

server 引数として、以下のプロパティを持つオブジェクトを渡します:

server	デフォルト値 (省略時)
.acceptUnsecureConnection : Boolean 暗号化されていない接続の確立が許可されてれば true	False
.accessTokenOAuth2: Text .accessTokenOAuth2: Object OAuth2 認証の資格情報を表すテキスト文字列またはトークンオブジェクト。 authenticationMode が OAUTH2 の場合のみ使用されます。 accessTokenOAuth2 が使用されているが authenticationMode が省略されていた場合、OAuth2 プロトコルが使用されます (サーバーで許可されなければ)。POP3 transporter オブジェクトには返されません。	なし
.authenticationMode : Text メールサーバーのセッションを開くのに使用される認証モード	サーバーがサポートするもっともセキュアな認証モードが使用されます
.connectionTimeOut : Integer サーバー接続の確立までに待機する最長時間 (秒単位)	30
.host : Text ホストサーバーの名前または IP アドレス	必須
.logFile : Text メール接続に対して定義された拡張ログファイル (あれば) へのフルパス	なし
.password : Text サーバーとの認証のためのユーザーパスワード POP3 transporter オブジェクトには返されません。	なし
.port : Integer メール通信に使用されるポート番号	995
.user : Text メールサーバーでの認証に使用されたユーザー名	なし

戻り値

この関数は、POP3 transporter オブジェクト を返します。返されるプロパティはすべて 読み取り専用 です。

POP3接続は、transporter オブジェクトが消去された時点で自動的に閉じられます。

例題

```
var $server : Object
$server:=New object
$server.host:="pop.gmail.com" // 必須
$server.port:=995
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXX"
$server.logFile:="LogTest.txt" // Logsフォルダーに保存するログ

var $transporter : 4D.POP3Transporter
$transporter:=POP3 New transporter($server)

$status:=$transporter.checkConnection()
If(Not($status.success))
    ALERT("メール受信中にエラーが発生しました: "+$status.statusText)
End if
```

4D.POP3Transporter.new()

4D.POP3Transporter.new(server : Object) : 4D.POP3Transporter

引数	タイプ		説明
server	Object	->	メールサーバー情報
戻り値	4D.POP3Transporter	<-	POP3 transporter オブジェクト

説明

4D.POP3Transporter.new() 関数は、新規の 4D.POP3Transporter 型オブジェクトを作成して返します。この関数の機能は、POP3 New transporter コマンドと同一です。

.acceptUnsecureConnection

▶履歴

.acceptUnsecureConnection : Boolean

説明

.acceptUnsecureConnection プロパティは、暗号化された接続が不可能な場合に、暗号化されていない接続の確立が許可されてれば true を格納します。

暗号化されていない接続が許可されていない場合には false が格納されており、その場合に暗号化された接続が不可能な場合にはエラーが返されます。

使用可能なセキュアなポートは次のとおりです:

- SMTP
 - 465: SMTPS
 - 587 または 25: STARTTLS アップグレードがされた SMTP (サーバーがサポートしていれば)
- IMAP
 - 143: IMAP 非暗号化ポート
 - 993: STARTTLS アップグレードがされた IMAP (サーバーがサポートしていれば)
- POP3
 - 110: POP3 非暗号化ポート
 - 995: STARTTLS アップグレードがされた POP3 (サーバーがサポートしていれば)

.authenticationMode

▶履歴

.authenticationMode : Text

説明

.authenticationMode プロパティは、メールサーバーのセッションを開くのに使用される認証モードを格納します。

デフォルトでは、サーバーによってサポートされている最も安全なモードが使用されます。

とりうる値:

値	定数	説明
APOP	POP3 authentication APOP	APOP プロトコルを使用した認証 (POP3 のみ)
CRAM-MD5	POP3 authentication CRAM-MD5	CRAM-MD5 プロトコルを使用した認証
LOGIN	POP3 authentication login	LOGIN プロトコルを使用した認証
OAuth2	POP3 authentication OAuth2	OAuth2 プロトコルを使用した認証
PLAIN	POP3 authentication plain	PLAIN プロトコルを使用した認証

.checkConnection()

▶履歴

.checkConnection() : Object

引数	タイプ	説明
戻り値	Object	<- transporter オブジェクト接続のステータス

説明

.checkConnection() 関数は、transporter オブジェクトが保存する情報を使用して接続をチェックします。必要なら再接続をし、そのステータスを返します。この関数を使用して、ユーザーから提供された値が有効かどうかを検証することができます。

返されるオブジェクト

この関数はメールサーバーにリクエストを送信し、メールステータスを表すオブジェクトを返します。このオブジェクトには、次のプロパティが格納されることがあります:

プロパティ		タイプ	説明
success		boolean	チェックが成功した場合には true、それ以外は false
status		number	(SMTPのみ) メールサーバーから返されたコード (メール処理に関係ない問題の場合には 0)
statusText		テキスト	メールサーバーから返されたステータスマッセージ、または 4Dエラースタック内に返された最後のエラー
errors		collection	4Dエラースタック (メールサーバーレスポンスが受信できた場合には返されません)
	[].errCode	number	4Dエラーコード
	[].message	テキスト	4Dエラーの詳細
	[].componentSignature	テキスト	エラーを返した内部コンポーネントの署名

例題

```

var $pw : Text
var $options : Object
$options:=New object

$pw:=Request("パスワードを入力してください:")
if(OK=1)
    $options.host:="pop3.gmail.com"

$options.user:="test@gmail.com"
$options.password:=$pw

$transporter:=POP3 New transporter($options)

$status:=$transporter.checkConnection()
If($status.success)
    ALERT("POP3接続チェックに成功しました。")
Else
    ALERT("Error: "+$status.statusText)
End if
End if

```

.connectionTimeOut

▶ 覆歴

.connectionTimeOut : Integer

説明

.connectionTimeOut プロパティは、サーバー接続の確立までに待機する最長時間 (秒単位)を格納します。 SMTP New transporter や POP3 New transporter 、 IMAP New transporter のコマンドで transporter オブジェクトを作成する際に使用される server オブジェクトにおいて、このプロパティが指定されなかった場合のデフォルトは 30 です。

.delete()

▶ 覆歴

.delete(msgNumber : Integer)

引数	タイプ		説明
msgNumber	Integer	->	削除するメッセージの番号

説明

.delete() 関数は、 msgNumber で指定したメールメッセージに対して、POP3サーバーから削除するためのフラグを立てます。

msgNumber には、削除するメールの番号を渡します。この番号は、 .getMailInfoList() 関数によって number プロパティに返されます。

この関数を実行しても、メールが実際に削除される訳ではありません。フラグが立てられたメールは、(POP3 New transporter で作成された) POP3_transporter オブジェクトが消去された時に初めて POP3サーバーから削除されます。立てたフラグは、 .undeleteAll() 関数を使用して削除することもできます。

カレントセッションが予期せず終了して接続が閉じられた場合（例：タイムアウト、ネットワーク問題等）にはエラーメッセージが生成され、削除フラグが立てられたメールは削除されずに POP3サーバー上に残ります。

例題

```

$mailInfoList:=$POP3_transporter.getMailInfoList()
For each($mailInfo;$mailInfoList)
    // "セッション終了時に削除" とメールのフラグを立てます
    $POP3_transporter.delete($mailInfo.number)
End for each
// セッションを強制的に終了し、削除フラグを立てたメールを削除します
CONFIRM("選択されているメッセージは削除されます。";"削除する";"元に戻す")
If(OK=1) // 削除を選んだ場合
    $POP3_transporter:=Null
Else
    $POP3_transporter.undeleteAll() // 削除フラグを消去します
End if

```

.getBoxInfo()

▶履歴

.getBoxInfo() : Object

引数	タイプ		説明
戻り値	Object	<-	boxInfo オブジェクト

説明

.getBoxInfo() 関数は、対象の `POP3 transporter` が指定するメールボックスに対応する `boxInfo` オブジェクトを返します。この関数を使用するとメールボックスに関する情報を取得することができます。

返される `boxInfo` オブジェクトには、以下のプロパティが格納されています:

プロパティ	タイプ	説明
mailCount	Number	メールボックス内のメッセージの数
size	Number	メッセージのサイズ (バイト単位)

例題

```

var $server; $boxinfo : Object

$server:=New object
$server.host:="pop.gmail.com" // 必須
$server.port:=995
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXX"

$transporter:=POP3 New transporter($server)

// メールボックス情報
$boxInfo:=$transporter.getBoxInfo()
ALERT("メールボックスには "+String($boxInfo.mailCount)+" 件のメッセージがあります。")

```

.getMail()

▶履歴

.getMail(`msgNumber` : Integer) : Object

引数	タイプ		説明
msgNumber	Integer	->	リスト中のメッセージの番号
戻り値	Object	<-	Email オブジェクト

説明

`.getMailInfo()` 関数は、`POP3 transporter` が指定するメールボックス内の、`msgNumber` に対応するメールの `mailInfo` オブジェクトを返します。この関数を使用すると、メールのコンテンツをローカルで管理できるようになります。

`msgNumber` には、取得するメッセージの番号を渡します。この番号は、`.getMailInfoList()` 関数によって `number` プロパティに返されます。

この関数は、以下の場合には Null を返します：

- `msgNumber` で指定したメッセージが存在しない場合
- 指定したメッセージが `.delete()` によって削除フラグが立てられていた場合

返されるオブジェクト

`.getMail()` は `Email オブジェクト` を返します。

例題

メールボックスにある最初のメールの送信者を調べます：

```
var $server; $transporter : Object
var $mailInfo : Collection
var $sender : Variant

$server:=New object
$server.host:="pop.gmail.com" // 必須
$server.port:=995
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXX"

$transporter:=POP3 New transporter($server)

$mailInfo:=$transporter.getMailInfoList()
$sender:=$transporter.getMail($mailInfo[0].number).from
```

.getMailInfo()

▶ 履歴

`.getMailInfo(msgNumber : Integer) : Object`

引数	タイプ		説明
msgNumber	Integer	->	リスト中のメッセージの番号
戻り値	Object	<-	MailInfo オブジェクト

説明

`.getMailInfoList()` 関数は、`POP3 transporter` が指定するメールボックス内の全メッセージについて記述した `mailInfo` オブジェクトのコレクションを返します。この関数を使用すると、POP3メールサーバー上にあるメッセージの一覧をローカルで管理することができるようになります。

`msgNumber` には、取得するメッセージの番号を渡します。この番号は、`.getMailInfoList()` 関数によって `number` プロパティに返されます。

返される `mailInfo` オブジェクトには、以下のプロパティが格納されています：

プロパティ	タイプ	説明
size	Number	メッセージのサイズ (バイト単位)
id	Text	メッセージの固有ID

この関数は、以下の場合には Null を返します：

- msgNumber で指定したメッセージが存在しない場合
- 指定したメッセージが `.delete()` によって削除フラグが立てられていた場合

例題

```

var $server; $mailInfo : Object
var $mailNumber : Integer

$server.host:="pop.gmail.com" // 必須
$server.port:=995
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXX"

var $transporter : 4D.POP3Transporter
$transporter:=POP3 New transporter($server)

// メッセージ情報
$mailInfo:=$transporter.getMailInfo(1) // 先頭メールを取得します
If($mailInfo #Null)
  ALERT("最初のメールのサイズは "+String($mailInfo.size)+" バイトです。")
End if

```

.getMailInfoList()

▶履歴

.getMailInfoList() : Collection

引数	タイプ	説明
戻り値	Collection	<- mailInfo オブジェクトのコレクション

説明

`POP3 New transporter` コマンドは、server 引数の指定に応じて新規の POP3接続を設定します。戻り値は、新しい `POP3 transporter` オブジェクトです。返された transporter オブジェクトは、通常メールの受信に使用されます。この関数を使用すると、POP3メールサーバー上にあるメッセージの一覧をローカルで管理することができるようになります。

返されるコレクションの各 `mailInfo` オブジェクトには、以下のプロパティが格納されています：

プロパティ	タイプ	説明
[].size	Number	メッセージのサイズ (バイト単位)
[].number	Number	メッセージの番号
[].id	Text	メッセージの固有ID (メッセージをローカルに保存する場合に有用です)

メールボックスにメッセージが一通もない場合、空のコレクションが返されます。

number と id プロパティについて

`number` プロパティは、`POP3_transporter` が作成された時点でのメールボックス内にあるメッセージの数です。`number` プロパティは、特定のメッセージと紐づいた静的な値ではなく、セッション開始時点のメールボックス内のメッセージ同士の関係に応じてセッション間で値が異なります。メッセージに割り当てられた番号は、`POP3_transporter` が維持されている間のみ有効です。`POP3_transporter` が消去されると、削除フラグが立てられていたメッセージは削除されます。ユーザーが再度サーバーにログインした場合、メールボックス内にあるカレントメッセージに対して、1 から x までの番号

が再度割り振られます。

これに対し、*id* プロパティは、メッセージがサーバーで受信された時に割り振られる固有の番号です。その番号はメッセージを受信した日付と時間を使用して計算され、POP3サーバーによって値が割り当てられます。残念ながら、POP3サーバーは *id* プロパティをメッセージに対する主な参照としては使用しません。POP3 セッションの間、サーバー上のメッセージを参照するには *number* プロパティを指定する必要があります。メッセージ参照をデータベース内に取得しながらメッセージ本文はサーバー上に残しておくようなソリューションを開発する場合、メッセージの指定には細心の注意を払う必要があります。

例題

メールボックス内にあるメールの総数と総サイズを取得します：

```
var $server : Object
$server:=New object
$server.host:="pop.gmail.com" // 必須
$server.port:=995
$server.user:="4d@gmail.com"
$server.password:="XXXXXXXX"

var $transporter : 4D.POP3Transporter
$transporter:=POP3 New transporter($server)

C_COLLECTION($mailInfo)
C_LONGINT($vNum;$vSize)

$mailInfo:=$transporter.getMailInfoList()
$vNum:=$mailInfo.length
$vSize:=$mailInfo.sum("size")

ALERT("メールボックスには "+String($vNum)+" 件のメッセージがあります。合計サイズは "+String($vSize)+" バイトです。")
```

.getMIMEAsBlob()

▶ 履歴

.getMIMEAsBlob(*msgNumber* : Integer) : Blob

引数	タイプ		説明
<i>msgNumber</i>	Integer	->	リスト中のメッセージの番号
戻り値	BLOB	<-	メールサーバーから返された MIME 文字列の BLOB

説明

.getMIMEAsBlob() 関数は、`POP3_transporter` が指定するメールボックス内の、*msgNumber* に対応するメッセージの MIME コンテンツを格納した BLOB を返します。

msgNumber には、取得するメッセージの番号を渡します。この番号は、`.getMailInfoList()` 関数によって *number* プロパティに返されます。

この関数は、以下の場合には空の BLOB を返します：

- *msgNumber* で指定したメッセージが存在しない場合
- 指定したメッセージが `.delete()` によって削除フラグが立てられていた場合

返される BLOB

`.getMIMEAsBlob()` は BLOB を返します。この BLOB はデータベースにアーカイブしたり、`MAIL Convert from MIME` コマンドを使用して `Email オブジェクト` へと変換したりすることができます。

例題

メールボックス内にあるメールの総数と総サイズを取得します：

```

var $server : Object
var $mailInfo : Collection
var $blob : Blob
var $transporter : 4D.POP3Transporter

$server:=New object
$server.host:="pop.gmail.com"
$server.port:=995
$server.user:="4d@gmail.com"
$server.password:="XXXXXXX"

$transporter:=POP3 New transporter($server)

$mailInfo:=$transporter.getMailInfoList()
$blob:=$transporter.getMIMEAsBlob($mailInfo[0].number)

```

.host

▶履歴

.host : Text

説明

.host プロパティは、ホストサーバーの名前または IPアドレスを格納します。この情報はメール通信 (SMTP、POP3、IMAP) に使用されます。

.logFile

▶履歴

.logFile : Text

説明

.logFile プロパティは、メール接続に対して定義された拡張ログファイル (あれば) へのフルパスを格納します。パスは、カレント Logs フォルダーを基準とした相対パス、あるいは絶対パスを指定できます。

`SET DATABASE PARAMETER` コマンドで有効化される通常のログファイルとは異なり、拡張ログファイルはすべての送信されたメールの MIMEコンテンツを保存し、サイズ制限がありません。拡張ログファイルの詳細については、以下の章をそれぞれ参照ください：

- SMTP 接続 - [4DSMTPLLog.txt](#)
- POP3 接続 - [4DPOP3Log.txt](#)
- IMAP 接続 - [4DIMAPLog.txt](#)

.port

▶履歴

.port : Integer

説明

.port プロパティは、メール通信に使用されるポート番号を格納します。 SMTP New transporter や POP3 New transporter、 IMAP New transporter のコマンドで transporter オブジェクトを作成する際に使用される server オブジェクトにおいて、このプロパティが指定されなかった場合に使用されるポートは次のとおりです：

- SMTP - 587
- POP3 - 995
- IMAP - 993

.undeleteAll()

▶ 覆歴

.undeleteAll() | 引数 | タイプ | | 説明 | | -- | --- | ::| ----- | | | | このコマンドは引数を必要としません |

説明

.undeleteAll() 関数は、POP3 transporter 内のメールに設定された削除フラグをすべて除去します。

.user

▶ 覆歴

.user : Text

説明

.user プロパティは、メールサーバーでの認証に使用されたユーザー名を格納します。

Session

プロジェクトにおいて、スケーラブルセッションが有効化されている場合、`Session` コマンドによって Session オブジェクトが返されます。Webクライアント（ブラウザなど）のセッションを制御するため、4D Webサーバーは自動的に Session オブジェクトを作成・管理します。このオブジェクトは、ユーザー セッションへのインターフェースを Web開発者に提供し、アクセス権の管理や、コンテキストデータの保存、プロセス間の情報共有、セッションに関連したプリエンプティブプロセスの開始などを可能にします。

セッションの実装に関する詳細については、[Webサーバーセッション](#) の章を参照ください。

概要

<code>.clearPrivileges()</code>	対象セッションに紐づいているアクセス権をすべて削除します
<code>.expirationDate : Text</code>	セッション cookie の有効期限
<code>.hasPrivilege(privilege : Text) : Boolean</code>	対象セッションに <i>privilege</i> のアクセス権が紐づいていれば true、でなければ false を返します
<code>.idleTimeout : Integer</code>	対象セッションが 4D によって終了されるまでの、非アクティブタイムアウト時間（分単位）
<code>.isGuest() : Boolean</code>	アクセス権のないゲストセッションの場合は true を返します
<code>.setPrivileges(privilege : Text)</code> <code>.setPrivileges(privileges : Collection)</code> <code>.setPrivileges(settings : Object)</code>	引数として渡したアクセス権をセッションと紐づけます
<code>.storage : Object</code>	Webクライアントのリクエストに対応するために情報を保存しておける共有オブジェクト
<code>.userName : Text</code>	セッションと紐づいたユーザー名

セッション

▶ 補足

Session : 4D.Session

引数	タイプ		説明
戻り値	4D.Session	<-	Session オブジェクト

説明

`Session` コマンドは、カレントのスケーラブルユーザー Web セッションに対応する `Session` オブジェクトを返します。

スケーラブルセッションが有効化されている 場合にのみ、このコマンドは機能します。セッションが無効な場合や、旧式セッションが使用されている場合には、`Null` を返します。

スケーラブルセッションが有効化されている場合、`Session` オブジェクトは次のコンテキストにおける、あらゆる Web プロセスから利用可能です：

- `On Web Authentication`、`On Web Connection`、および `On REST Authentication` データベースメソッド
- RESTリクエストで呼び出された ORDA データモデルクラス関数
- セミダイナミックページにおいて、4Dタグ (4DTEXT, 4DHML, 4DEVAL, 4DSRIPT/, 4DCODE) を介して処理されるコード
- "公開オプション: 4DタグとURL(4DACTION...)" を有効化されたうえで、4DACTION/ URL から呼び出されたプロジェクトメソッド

例題

"公開オプション: 4DタグとURL(4DACTION...)" を有効にした `action_Session` メソッドを定義しました。ブラウザーに次の URL を入力してメソッドを呼び出します:

```
IP:port/4DACTION/action_Session
```

```
//action_Session メソッド
Case of
  :(Session#Null)
    If(Session.hasPrivilege("WebAdmin")) // hasPrivilege 関数を呼び出します
      WEB SEND TEXT("4DACTION --> セッションは WebAdmin です")
    Else
      WEB SEND TEXT("4DACTION --> セッションは WebAdmin ではありません")
    End if
  Else
    WEB SEND TEXT("4DACTION --> セッションは null です")
End case
```

.clearPrivileges()

▶ 補足

`.clearPrivileges()` | 引数 | タイプ | | 説明 | | -- | --- | :: | ----- | | | | このコマンドは引数を必要としません |

説明

`.clearPrivileges()` 関数は、対象セッションに紐づいているアクセス権をすべて削除します。結果的に、当該セッションは自動的にゲストセッションになります。

例題

```
// セッションを無効にします
var $isGuest : Boolean

Session.clearPrivileges()
$isGuest:=Session.isGuest() // $isGuest は true
```

.expirationDate

▶ 補足

`.expirationDate` : Text

説明

`.expirationDate` プロパティは、セッションcookie の有効期限を返します。値は ISO 8601 標準に従って文字列で表現されます: YYYY-MM-DDTHH:MM:SS.mmmZ.

このプロパティは読み取り専用です。 `.idleTimeout` プロパティ値が変更された場合、有効期限は自動的に再計算されます。

例題

```
var $expiration : Text  
$expiration:=Session.expirationDate // 例: "2021-11-05T17:10:42Z"
```

.hasPrivilege()

▶ 覆歴

.hasPrivilege(*privilege* : Text) : Boolean

引数	タイプ		説明
privilege	Text	<-	確認するアクセス権の名称
戻り値	Boolean	<-	セッションが <i>privilege</i> のアクセス権を持っていれば true、それ以外は false

説明

.hasPrivilege() 関数は、対象セッションに *privilege* のアクセス権が紐づいていれば true、でなければ false を返します。

例題

"WebAdmin" アクセス権がセッションに紐づいているかを確認します:

```
If (Session.hasPrivilege("WebAdmin"))  
    // アクセス権が付与されているので、何もしません  
Else  
    // 認証ページを表示します  
  
End if
```

.idleTimeout

▶ 覆歴

.idleTimeout : Integer

説明

.idleTimeout プロパティは、対象セッションが 4D によって終了されるまでの、非アクティブタイムアウト時間 (分単位)を格納します。

プロパティ未設定時のデフォルト値は 60 (1時間) です。

このプロパティが設定されると、それに応じて .expirationDate プロパティも更新されます。

60 (分) 未満の値を指定することはできません (60 未満の値を設定した場合、タイムアウトは 60 (分) に設定されます)。

このプロパティは 読み書き可能 です。

例題

```
If (Session.isGuest())  
    // ゲストセッションは、60分の非アクティブ時間経過後に終了します  
    Session.idleTimeout:=60  
Else  
    // その他のセッションは、120分の非アクティブ時間経過後に終了します  
    Session.idleTimeout:=120  
End if
```

.isGuest()

▶ 履歴

.isGuest() : Boolean

引数	タイプ		説明
戻り値	Boolean	<-	ゲストセッションの場合は true、それ以外は false

説明

.isGuest() 関数は、アクセス権のないゲストセッションの場合は true を返します。

例題

On Web Connection データベースメソッドにて:

```
If (Session.isGuest())
    // ゲストユーザー用の処理
End if
```

.setPrivileges()

▶ 履歴

.setPrivileges(privilege : Text)
.setPrivileges(privileges : Collection)
.setPrivileges(settings : Object)

引数	タイプ		説明
privilege	Text	->	アクセス権の名称
privileges	Collection	->	アクセス権の名称のコレクション
settings	Object	->	"privileges" プロパティ (文字列またはコレクション) を持つオブジェクト

説明

.setPrivileges() 関数は、引数として渡したアクセス権をセッションと紐づけます。

- *privilege* には、アクセス権の名称を文字列として渡します (複数の場合はカンマ区切り)。
- *privileges* には、アクセス権の名称を文字列のコレクションとして渡します。
- *settings* には、以下のプロパティを持つオブジェクトを渡します:

プロパティ	タイプ	説明
privileges	Text または Collection	<ul style="list-style-type: none">● アクセス権名の文字列● アクセス権名のコレクション
userName	Text	(任意) セッションと紐づけるユーザー名

無効なアクセス権名を含む場合、*privileges* プロパティは無視されます。

現在の実装では、"WebAdmin" アクセス権のみ利用可能です。

セッションにアクセス権が紐づいていない場合、そのセッションはデフォルトで [ゲストセッション](#) です。

userName プロパティは Session オブジェクトレベルで利用可能で (読み取り専用)。

例題

カスタムな認証メソッドにおいて、ユーザーに "WebAdmin" アクセス権を付与します:

```
var $userOK : Boolean  
... // ユーザー認証  
  
If ($userOK) // ユーザー認証に成功した場合  
    var $info : Object  
    $info:=New object()  
    $info.privileges:=New collection("WebAdmin")  
    Session.setPrivileges($info)  
End if
```

.storage

▶ 補足

.storage : Object

説明

.storage プロパティは、Webクライアントのリクエストに対応するために情報を保存しておける共有オブジェクトを格納します。

Session オブジェクトの作成時には、.storage プロパティは空です。共有オブジェクトのため、このプロパティはサーバー上の Storage オブジェクトにおいて利用可能です。

サーバーの Storage オブジェクトと同様に、.storage プロパティは常に "single" で存在します。共有オブジェクトや共有コレクションを .storage に追加しても、共有グループは作成されません。

このプロパティは 読み取り専用 ですが、戻り値のオブジェクトは読み書き可能です。

例題

クライアントの IP を .storage プロパティに保存します。 On Web Authentication データベースメソッドに以下のように書けます:

```
If (Session.storage.clientIP=NULL) // 最初のアクセス  
    Use (Session.storage)  
        Session.storage.clientIP:=New shared object("value"; $clientIP)  
    End use  
End if
```

.userName

▶ 補足

.userName : Text

説明

.userName プロパティは、セッションと紐づいたユーザー名を格納します。このプロパティは、コード内でユーザーを確認するのに使用できます。

このプロパティはデフォルトでは空の文字列です。これは、`setPrivileges()` 関数の `privileges` プロパティを使って設定することができます。

このプロパティは 読み取り専用 です。

Signal

シグナルは、マルチプロセスアプリケーションにおいてプロセス間でのやり取りを管理し衝突を避けるために 4Dランゲージが提供するツールです。シグナルは、1つ以上のプロセスが実行を一時停止し、特定のタスクが完了するまで待つようにする仕組みです。どのプロセスもシグナルを待機またはリリースすることができます。

プロセス間のやり取りを管理するには、セマフォーも使用できます。セマフォーは、2つ以上のプロセスが同じリソース（ファイル、レコードなど）を同時に変更しないようにするための仕組みです。セマフォーを解除できるのは、それを設定したプロセスのみです。

Signal オブジェクト

シグナルは、ワーカーやプロセスを呼び出す/作成するコマンドに対して引数として渡す必要のある共有オブジェクトです。

4D Signal オブジェクトは次のビルトインされたメソッドおよびプロパティを持ちます：

- `.wait()`
- `.trigger()`
- `.signaled`
- `.description`

シグナルの `.wait()` メソッドを呼び出したワーカーやプロセスは、`.signaled` プロパティが `true` になるまで実行を停止します。シグナルを待っている間、呼び出したプロセスは CPU を消費しません。これはマルチプロセスアプリケーションのパフォーマンスを鑑みると有意義な仕組みです。`.signaled` プロパティは、いずれかのワーカーまたはプロセスがシグナルの `.trigger()` メソッドを呼び出した時点で `true` になります。

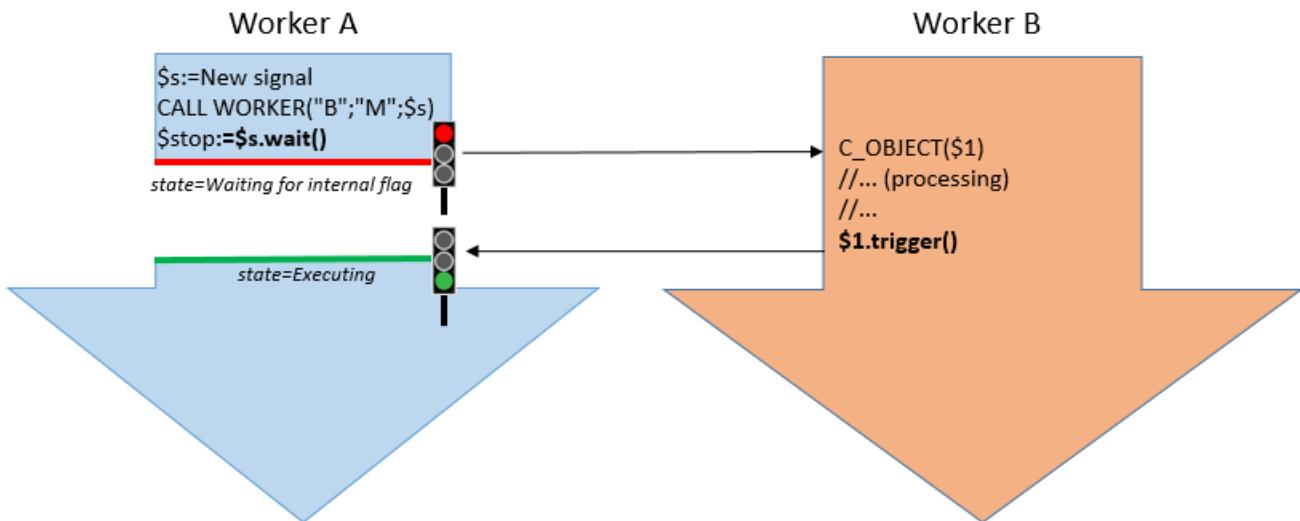
また、コードのブロックを避けるため、`.wait()` 呼び出しの際に定義したタイムアウト時間に達することでも待機状態を脱することができます。

Signal オブジェクトは、[New signal](#) コマンドによって作成されます。

シグナルの使い方

4D では、[New signal](#) コマンドを呼び出すことで新規 Signal オブジェクトを作成します。作成したシグナルは、[New process](#) あるいは `CALL WORKER` コマンドに引数として渡す必要があります。それにより、プロセスやワーカーはタスクを完了した際にシグナルを書き換えることができます。

- `signal.wait()` は、他のワーカー/プロセスのタスクが完了するまで待機するワーカー/プロセスから呼び出す必要があります。
- `signal.trigger()` は、他のワーカー/プロセスを待機状態から解放するために、タスク実行を終えたワーカー/プロセスが呼び出す必要があります。



`signal.trigger()` の呼び出しによって解放されたシグナルは、再利用することができません。別のシグナルを設定するには、[New signal](#) コマンドをあらためて呼び出す必要があります。

Signal オブジェクトは [共有オブジェクト](#) であるため、呼び出されたワーカー/プロセスから結果を返すために使用することもできます。この場合、`Use...End use` 構文内で値を書き込む必要があります。

例題

```
var $signal : 4D.Signal

// シグナルを作成します
$signal:=New signal

// メインプロセスを呼び出し、OpenForm メソッドを実行します
CALL WORKER(1;"OpenForm";$signal)
// 他の計算をします
...
// プロセスの終了を待機します
$signaled:=$signal.wait()

// 結果を処理します
$calc:=$signal.result+...
```

OpenForm メソッド:

```
#DECLARE ($signal : 4D.Signal)
var $form : Object
$form:=New object("value";0)

// フォームを開きます
$win:=Open form window("Information";Movable form dialog box)
DIALOG("Information";$form)
CLOSE WINDOW($win)

// $signal 共有オブジェクトに新しい属性を追加することで、他のプロセスに返り値を受け渡します：
Use($signal)
  $signal.result:=$form.value
End use

// 待機プロセスに向けてシグナルをトリガーします
$signal.trigger()
```

概要

.description : Text
Signal オブジェクトのカスタムな詳細

.signaled : Boolean
Signal オブジェクトの現在の状態

.trigger()
シグナルオブジェクトの signaled プロパティを true に設定します

.wait({ timeout : Real }) : Boolean
シグナルオブジェクトの .signaled プロパティが true になるか、任意の timeout に指定したタイムアウト時間が経過するまで、カレントプロセスを待機させます

New signal

▶ 覆歴

New signal { (description : Text) } : 4D.Signal

引数	タイプ		説明
description	Text	->	シグナルの詳細
戻り値	4D.Signal	<-	シグナルを格納するネイティブオブジェクト

説明

`New signal` コマンドは、`4D.Signal` オブジェクトを作成します。

シグナルは、ワーカー/プロセスから他のワーカー/プロセスへと引数のように渡せる共有オブジェクトです。そのため、以下のようなことが可能になります：

- 呼び出されたワーカー/プロセスは特定の処理が完了した後に Signal オブジェクトを更新することができます。
- 呼び出し元のワーカー/プロセスは CPUリソースを消費することなく、実行を停止してシグナルが更新されるまで待つことができます。

任意で `description` 引数に、シグナルの詳細を説明するカスタムテキストを渡すことができます。テキストは、シグナルの作成後に定義することも可能です。

Signal オブジェクトは共有オブジェクトのため、`Use...End use` 構文を使用することで、`.description` プロパティのほか、ユーザー独自のプロパティを管理するのに使用することもできます。

戻り値

新規の `4D.Signal` オブジェクト。

例題

以下は、シグナルを設定するワーカーの典型的な例です：

```
var $signal : 4D.Signal
$signal:=New signal("This is my first signal")

CALL WORKER("myworker";"doSomething";$signal)
$signaled:=$signal.wait(1) // 最大で1秒待機します

If($signaled)
    ALERT("myworker はタスクを終了しました。 結果: "+$signal.myresult)
Else
    ALERT("myworker は 1秒以内にタスクを終了できませんでした。")
End if
```

以下は、`doSomething` メソッドの一例です：

```
#DECLARE ($signal : 4D.Signal)
// 何らかの処理
// ...
Use($signal)
    $signal.myresult:=$processingResult // 結果を返します
End use
$signal.trigger() // 処理が完了しました
```

.description

▶履歴

`.description` : Text

説明

`.description` プロパティは、`Signal` オブジェクトのカスタムな詳細を格納します。

`.description` は、`Signal` オブジェクトの作成時、あるいはその他のタイミングでも設定することができます。ただし、`Signal` オブジェクトは共有オ

プロジェクトであるため、`.description` プロパティに書き込む際には必ず `Use...End use` 構文を使わなくてはならない点に留意が必要です。

読み書き可能 プロパティです。

.signaled

▶ 履歴

`.signaled` : Boolean

説明

`.signaled` プロパティは、`Signal` オブジェクトの現在の状態を格納します。`Signal` オブジェクトが作成された時点では、`.signaled` は `false` です。`Signal` オブジェクトに対して `.trigger()` が呼び出された時に `true` となります。

このプロパティは 読み取り専用 です。

.trigger()

▶ 履歴

`.trigger()` | 引数 | タイプ | | 説明 | | -- | --- | ::| ----- | | | | このコマンドは引数を必要としません |

説明

`.trigger()` 関数は、シグナルオブジェクトの `signaled` プロパティを `true` に設定します。すると、このシグナルを待機していたワーカーやプロセスが開始されます。

`Signal` がすでにシグナルされている（つまり `signaled` プロパティが `true` になっている）状態であった場合、この関数は何もしません。

.wait()

▶ 履歴

`.wait({ timeout : Real })` : Boolean

引数	タイプ		説明
timeout	Real	->	シグナルの最大待機時間 (秒単位)
戻り値	Boolean	<-	<code>.signaled</code> プロパティの状態

説明

`.wait()` 関数は、シグナルオブジェクトの `.signaled` プロパティが `true` になるか、任意の `timeout` に指定したタイムアウト時間が経過するまで、カレントプロセスを待機させます。

コード実行のブロックを防ぐため、`timeout` 引数を使用して最長待機時間を秒単位で指定することもできます（小数を使用できます）。

警告: `timeout` 引数を渡さずに `.wait()` を 4D のメインプロセスで呼び出すことは推奨されていません。最悪の場合 4D アプリケーション全体がフリーズしてしまう恐れがあります。

`Signal` がすでにシグナルされている（つまり `signaled` プロパティが `true` になっている）状態であった場合、この関数は即座に戻り値を返します。

この関数は `.signaled` プロパティの値を返します。この値を評価することで、待機が終了したのは `.trigger()` が呼び出されたためか（`.signaled` プロパティは `true`）、それともタイムアウト時間が経過したためか（`.signaled` プロパティは `false`）を知ることができます。

Signal オブジェクトを待機しているプロセスの状態は `Waiting for internal flag` です。

SMTPTransporter

`SMTPTransporter` クラスを使って、SMTP接続の設定や、*SMTP transporter* オブジェクトを介したメールの送信をおこなうことができます。

SMTP Transporter オブジェクト

SMTP Transporter オブジェクトは [SMTP New transporter](#) コマンドによってインスタンス化されます。これらは、次のプロパティや関数を持ちます：

`.acceptUnsecureConnection : Boolean`

暗号化されていない接続の確立が許可されれば `true`

`.authenticationMode : Text`

メールサーバーのセッションを開くのに使用される認証モード

`.bodyCharset : Text`

メール本文で使用される文字セットとエンコーディング

`.checkConnection() : Object`

`transporter` オブジェクトが保存する情報を使用して接続をチェックします

`.connectionTimeOut : Integer`

サーバー接続の確立までに待機する最長時間 (秒単位)

`.headerCharset : Text`

メールヘッダーで使用される文字セットとエンコーディング

`.host : Text`

ホストサーバーの名前または IPアドレス

`.keepAlive : Boolean`

`transporter` オブジェクトが抹消されるまで、SMTP接続が維持されなければならない場合に `true`

`.logFile : Text`

メール接続に対して定義された拡張ログファイル (あれば) へのフルパス

`.port : Integer`

メール通信に使用されるポート番号

`.send(mail : Object) : Object`

`mail` 引数が指定するメールメッセージを、`transporter` オブジェクトが定義する SMTPサーバーへと送信し、ステータスオブジェクトを返します

`.sendTimeOut : Integer`

`.send()` 呼び出し時のタイムアウト時間 (秒単位)

`.user : Text`

メールサーバーでの認証に使用されたユーザー名

SMTP New transporter

▶ 補足

`SMTP New transporter(server : Object) : 4D.SMTPTransporter`

引数	タイプ		説明
server	Object	->	メールサーバー情報
戻り値	4D.SMTPTransporter	<-	SMTP transporter オブジェクト

説明

`SMTP New transporter` コマンドは、`server` 引数の指定に応じて 新規の SMTP接続を設定します。戻り値は、新しい *SMTP transporter* オブジェクトです。返された transporter オブジェクトは、通常メールの送信に使用されます。

このコマンドは SMTPサーバーとの接続を開始しません。SMTP接続は、実際には `.send()` 関数が実行された時に開かれます。

SMTP接続は、以下の場合に自動的に閉じられます：

* `keepAlive` プロパティが `true` (デフォルト) の場合には、transporter オブジェクトが消去された時。
* `keepAlive` プロパティが `false` の場合には、各 `.send()` 関数が実行された後。

`server` 引数として、以下のプロパティを持つオブジェクトを渡します：

<code>server</code>		デフォルト値 (省略時)
<code>.acceptUnsecureConnection : Boolean</code> 暗号化されていない接続の確立が許可されれば true	False	
<code>.accessTokenOAuth2 : Text</code> <code>.accessTokenOAuth2 : Object</code> OAuth2 認証の資格情報を表すテキスト文字列またはトークンオブジェクト。 <code>authenticationMode</code> が OAUTH2 の場合のみ使用されます。 <code>accessTokenOAuth2</code> が使用されているが <code>authenticationMode</code> が省略された場合、OAuth2 プロトコルが使用されます (サーバーで許可されていれば)。 <i>SMTP transporter</i> オブジェクトには返されません。	なし	
<code>.authenticationMode : Text</code> メールサーバーのセッションを開くのに使用される認証モード		サーバーがサポートするもつともセキュアな認証モードが使用されます
<code>.bodyCharset : Text</code> メール本文で使用される文字セットとエンコーディング		<code>mail mode UTF8 (US-ASCII_UTF8_QP)</code>
<code>.connectionTimeOut : Integer</code> サーバー接続の確立までに待機する最長時間 (秒単位)	30	
<code>.headerCharset : Text</code> メールヘッダーで使用される文字セットとエンコーディング		<code>mail mode UTF8 (US-ASCII_UTF8_QP)</code>
<code>.host : Text</code> ホストサーバーの名前または IP アドレス		必須
<code>.keepAlive : Boolean</code> <code>transporter</code> オブジェクトが抹消されるまで、SMTP 接続が維持されなければならない場合に true	True	
<code>.logFile : Text</code> メール接続に対して定義された拡張ログファイル (あれば) へのフルパス	なし	
<code>password : Text</code> サーバーとの認証のためのユーザーパスワード <i>SMTP transporter</i> オブジェクトには返されません。	なし	
<code>.port : Integer</code> メール通信に使用されるポート番号	587	
<code>.sendTimeOut : Integer</code> <code>.send()</code> 呼び出し時のタイムアウト時間 (秒単位)	100	
<code>.user : Text</code> メールサーバーでの認証に使用されたユーザー名	なし	

戻り値

この関数は、*SMTP transporter* オブジェクト を返します。返されるプロパティはすべて 読み取り専用 です。

例題

```

$server:=New object
$server.host:="smtp.gmail.com" // 必須
$server.port:=465
$server.user:="4D@gmail.com"
$server.password:="XXXX"
$server.logFile:="LogTest.txt" // Logsフォルダーに保存する拡張されたログ

var $transporter : 4D.SMTPTransporter
$transporter:=SMTP New transporter($server)

$email:=New object
$email.subject:="my first mail "
$email.from:="4d@gmail.com"
$email.to:="4d@4d.com;test@4d.com"
$email.textBody:="Hello World"
$email.htmlBody:="

# Hello World



#### 'Neque porro quisquam est qui dolorem ipsum quia dolor sit am <p>There are many variations of passages of Lorem Ipsum available."\ +"The generated Lorem Ipsum is therefore always free from repetition, injected humour, or non-character $status:=$transporter.send($email) If(Not($status.success)) ALERT("メール送信中にエラーが発生しました: "+$status.message) End if


```

4D.SMTPTransporter.new()

4D.SMTPTransporter.new(*server* : Object) : 4D.SMTPTransporter

引数	タイプ		説明
server	Object	->	メールサーバー情報
戻り値	4D.SMTPTransporter	<-	SMTP transporter オブジェクト

説明

4D.SMTPTransporter.new() 関数は、新規の 4D.SMTPTransporter 型オブジェクトを作成して返します。この関数の機能は、 SMTP New transporter コマンドと同一です。

.acceptUnsecureConnection

▶履歴

.acceptUnsecureConnection : Boolean

説明

.acceptUnsecureConnection プロパティは、暗号化された接続が不可能な場合に、暗号化されていない接続の確立が許可されてれば true を格納します。

暗号化されていない接続が許可されていない場合には false が格納されており、その場合に暗号化された接続が不可能な場合にはエラーが返されます。

使用可能なセキュアなポートは次のとおりです:

- SMTP
 - 465: SMTPS
 - 587 または 25: STARTTLS アップグレードがされた SMTP (サーバーがサポートしていれば)
- IMAP

- 143: IMAP 非暗号化ポート
- 993: STARTTLS アップグレードがされた IMAP (サーバーがサポートしていれば)
- POP3
 - 110: POP3 非暗号化ポート
 - 995: STARTTLS アップグレードがされた POP3 (サーバーがサポートしていれば)

.authenticationMode

▶履歴

.authenticationMode : Text

説明

.authenticationMode プロパティは、メールサーバーのセッションを開くのに使用される認証モードを格納します。

デフォルトでは、サーバーによってサポートされている最も安全なモードが使用されます。

とりうる値:

値	定数	説明
CRAM-MD5	SMTP authentication CRAM MD5	CRAM-MD5 プロトコルを使用した認証
LOGIN	SMTP authentication login	LOGIN プロトコルを使用した認証
OAuth2	SMTP authentication OAuth2	OAuth2 プロトコルを使用した認証
PLAIN	SMTP authentication plain	PLAIN プロトコルを使用した認証

.bodyCharset

▶履歴

.bodyCharset : Text

説明

.bodyCharset プロパティは、メール本文で使用される文字セットとエンコーディングを格納します。

- 件名
- 添付ファイル名
- メール名

とりうる値:

定数	値	説明
mail mode ISO2022JP	US-ASCII_ISO-2022-JP_UTF8_QP	<ul style="list-style-type: none"> • <i>headerCharset</i>: 可能なら US-ASCII、次に可能なら Japanese (ISO-2022-JP) & Quoted-printable、それも不可なら UTF-8 & Quoted-printable • <i>bodyCharset</i>: 可能なら US-ASCII、次に可能なら Japanese (ISO-2022-JP) & 7-bit、それも不可なら UTF-8 & Quoted-printable
mail mode ISO88591	ISO-8859-1	<ul style="list-style-type: none"> • <i>headerCharset</i>: ISO-8859-1 & Quoted-printable • <i>bodyCharset</i>: ISO-8859-1 & 8-bit
mail mode UTF8	US-ASCII_UTF8_QP	<i>headerCharset</i> & <i>bodyCharset</i> : 可能なら US-ASCII、それが不可なら UTF-8 & Quoted-printable (デフォルト値)
mail mode UTF8 in base64	US-ASCII_UTF8_B64	<i>headerCharset</i> & <i>bodyCharset</i> : 可能な場合は US-ASCII、それ以外は UTF-8 & base64

.checkConnection()

▶履歴

.checkConnection() : Object

引数	タイプ		説明
戻り値	Object	<-	transporter オブジェクト接続のステータス

説明

.checkConnection() 関数は、transporter オブジェクトが保存する情報を使用して接続をチェックします。必要なら再接続をし、そのステータスを返します。この関数を使用して、ユーザーから提供された値が有効かどうかを検証することができます。

返されるオブジェクト

この関数はメールサーバーにリクエストを送信し、メールステータスを表すオブジェクトを返します。このオブジェクトには、次のプロパティが格納があります：

プロパティ		タイプ	説明
success		boolean	チェックが成功した場合には true、それ以外は false
status		number	(SMTPのみ) メールサーバーから返されたコード (メール処理に関係ない問題の場合には 0)
statusText		テキスト	メールサーバーから返されたステータスマッセージ、または 4Dエラースタック内に返された最後のエラー
errors		collection	4Dエラースタック (メールサーバーレスポンスが受信できた場合には返されません)
	[].errCode	number	4Dエラーコード
	[].message	テキスト	4Dエラーの詳細
	[].componentSignature	テキスト	エラーを返した内部コンポーネントの署名

SMTPステータスコードについての詳細は [こちらのページ](#) を参照ください。

例題

```
var $pw : Text
var $options : Object
var $transporter : 4D.SMTPTransporter
$options:=New object

$pw:=Request("パスワードを入力してください:")
$options.host:="smtp.gmail.com"

$options.user:="test@gmail.com"
$options.password:=$pw

$transporter:=SMTP New transporter($options)

$status:=$transporter.checkConnection()
If($status.success=True)
    ALERT("SMTP接続チェックに成功しました。")
Else
    ALERT("エラー # "+String($status.status)+", "+$status.statusText)
End if
```

.connectionTimeOut

▶ 補足

.connectionTimeOut : Integer

説明

.connectionTimeOut プロパティは、サーバー接続の確立までに待機する最長時間(秒単位)を格納します。SMTP New transporter や POP3 New transporter、IMAP New transporter のコマンドで transporter オブジェクトを作成する際に使用される server オブジェクトにおいて、このプロパティが指定されなかった場合のデフォルトは 30 です。

.headerCharset

▶ 補足

.headerCharset : Text

説明

.headerCharset プロパティは、メールヘッダーで使用される文字セットとエンコーディングを格納します。ヘッダーにはメールの次の要素を含みます：

- 件名
- 添付ファイル名
- メール名

とりうる値：

定数	値	説明
mail mode ISO2022JP	US-ASCII_ISO-2022-JP_UTF8_QP	<ul style="list-style-type: none">• headerCharset: 可能なら US-ASCII、次に可能なら Japanese (ISO-2022-JP) & Quoted-printable、それも不可なら UTF-8 & Quoted-printable• bodyCharset: 可能なら US-ASCII、次に可能なら Japanese (ISO-2022-JP) & 7-bit、それも不可なら UTF-8 & Quoted-printable
mail mode ISO88591	ISO-8859-1	<ul style="list-style-type: none">• headerCharset: ISO-8859-1 & Quoted-printable• bodyCharset: ISO-8859-1 & 8-bit
mail mode UTF8	US-ASCII_UTF8_QP	headerCharset & bodyCharset: 可能なら US-ASCII、それが不可なら UTF-8 & Quoted-printable (デフォルト値)
mail mode UTF8 in base64	US-ASCII_UTF8_B64	headerCharset & bodyCharset: 可能な場合は US-ASCII、それ以外は UTF-8 & base64

.host

▶ 補足

.host : Text

説明

.host プロパティは、ホストサーバーの名前または IPアドレスを格納します。この情報はメール通信(SMTP、POP3、IMAP)に使用されます。

.keepAlive

▶ 補足

.keepAlive : Boolean

説明

.keepAlive プロパティは、transporter オブジェクトが抹消されるまで、SMTP接続が維持されなければならない場合に true、それ以外は False を格納します。SMTP New transporter コマンドで transporter オブジェクトを作成する際に使用する server オブジェクトにおいて、keepAlive プロパティが指定されなかった場合のデフォルトは true です。

SMTP接続は、以下の場合に自動的に閉じられます:

- `.keepAlive` プロパティが `true` (デフォルト) の場合には、`transporter` オブジェクトが消去された時。
- `.keepAlive` プロパティが `false` の場合には、各 `.send()` 関数が実行された後。

.logFile

▶履歴

`.logFile` : Text

説明

`.logFile` プロパティは、メール接続に対して定義された拡張ログファイル (あれば) へのフルパスを格納します。パスは、カレント Logs フォルダーを基準とした相対パス、あるいは絶対パスを指定できます。

`SET DATABASE PARAMETER` コマンドで有効化される通常のログファイルとは異なり、拡張ログファイルはすべての送信されたメールの MIMEコンテンツを保存し、サイズ制限がありません。拡張ログファイルの詳細については、以下の章をそれぞれ参照ください:

- SMTP 接続 - [4DSMTPLLog.txt](#)
- POP3 接続 - [4DPOP3Log.txt](#)
- IMAP 接続 - [4DIMAPLog.txt](#)

.port

▶履歴

`.port` : Integer

説明

`.port` プロパティは、メール通信に使用されるポート番号を格納します。`SMTP New transporter` や `POP3 New transporter`、`IMAP New transporter` のコマンドで `transporter` オブジェクトを作成する際に使用される `server` オブジェクトにおいて、このプロパティが指定されなかった場合に使用されるポートは次のとおりです:

- SMTP - 587
- POP3 - 995
- IMAP - 993

.send()

▶履歴

`.send(mail : Object) : Object`

引数	タイプ		説明
mail	Object	->	送信する メール
戻り値	Object	<-	SMTP ステータス

説明

`.send()` 関数は、`mail` 引数が指定するメールメッセージを、`transporter` オブジェクトが定義する SMTPサーバーへと送信し、ステータスオブジェクトを返します。

`transporter` オブジェクトは、事前に `SMTP New transporter` コマンドによって作成されている必要があります。

この関数は、SMTP接続が事前に開かれていない場合は、それを作成します。`transporter` オブジェクトの `.keepAlive` プロパティが `false` であった場合、SMTP接続は `.send()` 実行後に自動的に閉じられます。詳細については、`SMTP New transporter` コマンドの説明を参照してください。

`mail`には、送信する有効な `Email` オブジェクトを渡します。メールには送信元 (メールがどこから送られるか) と送信先 (一名以上の受信者) プロ

パーティが含まれている必要がありますが、その他のプロパティは任意です。

返されるオブジェクト

この関数は、SMTP ステータスを表すオブジェクトを返します。このオブジェクトには、次のプロパティが格納されることがあります：

プロパティ	タイプ	説明
success	boolean	送信に成功した場合は true、それ以外は false
status	number	SMTPサーバーから返されたコード（メール処理に関係ない問題の場合には 0）
statusText	テキスト	SMTPから返されるステータスマッセージ

SMTP 処理とは関係のない問題（例：必須プロパティがメールにない）が発生した場合、4D はエラーを生成します。これは、`ON ERR CALL` コマンドでインストールしたメソッドでインターセプトできます。エラー情報を取得するには、`GET LAST ERROR STACK` コマンドを使用します。

この場合、結果のステータスオブジェクトには以下の値が含まれます：

プロパティ	値
success	False
status	0
statusText	"Failed to send email"

.sendTimeOut

▶ 履歴

`.sendTimeOut : Integer`

説明

`.sendTimeOut` プロパティは、`.send()` 呼び出し時のタイムアウト時間（秒単位）を格納します。`.sendTimeOut` プロパティが `server` オブジェクトによって設定されていない場合は、デフォルトで 100 という値が使用されます。

.user

▶ 履歴

`.user : Text`

説明

`.user` プロパティは、メールサーバーでの認証に使用されたユーザー名を格納します。

SystemWorker

システムワーカーを使うことで、4Dコードは同じマシン上で任意の外部プロセス（シェルコマンド、PHPなど）を呼び出すことができます。システムワーカーは非同期で呼び出されます。コールバックを使用することで、4Dは双方向の通信を可能にします。

`SystemWorker` クラスは、`4D` クラスストアにて提供されています。

例題

```
// ipconfig 情報へのアクセスを取得する Windows での例
var $myWinWorker : 4D.SystemWorker
var $ipConfig : Text
$myWinWorker:= 4D.SystemWorker.new("ipconfig")
$ipConfig:=$myWinWorker.wait(1).response //timeout 1 second

// ファイルのパーミッションを変更する macOS での例
// chmod はパーミッションを変更するための macOS コマンドです
var $myMacWorker : 4D.SystemWorker
$myMacWorker:= 4D.SystemWorker.new("chmod +x /folder/myfile.sh")
```

概要

4D.SystemWorker.new (*commandLine* : Text { ; *options* : Object }) : 4D.SystemWorker

外部プロセスを開始するために *commandLine* に渡したコマンドラインを実行する `4D.SystemWorker` オブジェクトを作成し、返します

.closeInput()

外部プロセスの入力ストリーム (*stdin*) を閉じます

.commandLine : Text

`new()` 関数に引数として渡したコマンドライン

.currentDirectory : 4D.Folder

外部プロセスが実行される作業ディレクトリ

.dataType : Text

レスポンス本文のデータ型

.encoding : Text

レスポンス本文のエンコーディング

.errors : Collection

実行エラーの場合、4Dエラーのコレクション

.exitCode : Integer

外部プロセスから返される終了コード

.hideWindow : Boolean

実行ファイルや DOSコンソールのウィンドウを隠すのに使用できます (Windows のみ)

.pid : Integer

外部プロセスの、システムレベルでの一意的な識別子

.postMessage(*message* : Text)

.postMessage(*messageBLOB* : Blob)

外部プロセスの入力ストリーム (*stdin*) への書き込みをおこないます

.response : Text

.response : Blob

リクエストが終了した時点で、返された全データの結合

.responseError : Text

リクエストが終了した時点で、返された全エラーの結合

.terminate()

`SystemWorker` の実行を強制終了します

.terminated : Boolean

外部プロセスが終了された場合に true

.timeout : Integer

外部プロセスが生きている場合、キルされるまでの秒数

.wait({*timeout* : Real}) : 4D.SystemWorker

`SystemWorker` の実行終了まで、または *timeout* で指定した時間が経過するまで待機します

4D.SystemWorker.new()

▶履歴

4D.SystemWorker.new (*commandLine* : Text { ; *options* : Object }) : 4D.SystemWorker

引数	タイプ		説明
<i>commandLine</i>	テキスト	->	実行するコマンドライン
<i>options</i>	オブジェクト	->	ワーカーパラメーター
<i>result</i>	4D.SystemWorker	<-	非同期の新規システムワーカー (プロセスが開始されなかった場合は null)

説明

`4D.SystemWorker.new()` 関数は、外部プロセスを開始するために *commandLine* に渡したコマンドラインを実行する 4D.SystemWorker オブジェクトを作成し、返します。

返されたシステムワーカーオブジェクトは、ワーカーにメッセージを送信したり、ワーカーの結果を取得するために使用できます。

プロキシオブジェクトの生成中に問題があった場合、この関数は `null` オブジェクトを返し、エラーが生成されます。

commandLine には、実行するアプリケーションのファイルのフルパス (POSIX シンタックス)、および必要に応じて追加の引数を渡します。アプリケーション名だけを渡すと、4Dは 実行ファイルを探すために `PATH` 環境変数を使用します。

警告: この関数は、実行可能なアプリケーションを起動するだけで、シェル (コマンドインタープリター) の一部である命令を実行することはできません。たとえば Windows で、このコマンドを使用して `dir` 命令を実行することはできません。

options オブジェクト

options に渡すオブジェクトは、次のプロパティを持つことができます:

プロパティ	タイプ	デフォルト	説明
<i>onResponse</i>	Formula	未定義	システムワーカーメッセージ用のコールバック。完全なレスポンスを受け取り次第、このコールバックが呼び出されます。コールバックは 2つのオブジェクトを引数として受け取ります (後述参照)
<i>onData</i>	Formula	未定義	システムワーカーデータ用のコールバック。システムワーカーがデータを受け取る度に、このコールバックが呼び出されます。コールバックは 2つのオブジェクトを引数として受け取ります (後述参照)
<i>onDataError</i>	Formula	未定義	外部プロセスエラー用のコールバック (外部プロセスの <code>stderr</code>)。コールバックは 2つのオブジェクトを引数として受け取ります (後述参照)
<i>onError</i>	Formula	未定義	外部プロセスが終了されたときのコールバック。コールバックは 2つのオブジェクトを引数として受け取ります (後述参照)
<i>onTerminate</i>	Formula	未定義	外部プロセスが終了されたときのコールバック。コールバックは 2つのオブジェクトを引数として受け取ります (後述参照)
<i>timeout</i>	数値	未定義	プロセスが生きている場合、キルされるまでの秒数。
<i>dataType</i>	テキスト	"text"	レスポンス本文のデータ型。可能な値: "text" (デフォルト), "blob"。
<i>encoding</i>	テキスト	"UTF-8"	<code>dataType="text"</code> の場合のみ。レスポンス本文のエンコーディング。利用可能な値については、 CONVERT FROM TEXT コマンドの説明を参照ください。
<i>variables</i>	オブジェクト		システムワーカー用のカスタム環境変数を設定します。シンタックス: <code>variables.key=value</code> (key は変数名、value はその値)。値は、可能な限り文字列に変換されます。値に '=' を含めることはできません。定義されていない場合、システムワーカーは 4D環境を継承します。
<i>currentDirectory</i>	Folder		プロセスが実行される作業ディレクトリ
<i>hideWindow</i>	ブール	true	(Windows) アプリケーションウィンドウを隠す (可能な場合)、または Windowsコンソールを隠す

すべてのコールバック関数は、2つのオブジェクト引数を受け取ります。その内容は、コールバックに依存します:

引数	タイプ	onResponse	onData	onDataError	onError	onTerminate
\$param1	オブジェクト	SystemWorker	SystemWorker	SystemWorker	SystemWorker	SystemWorker
\$param2.type	テキスト	"response"	"data"	"error"	"error"	"termination"
\$param2.data	Text または Blob		取得データ	エラーデータ		

以下は、コールバック呼び出しの流れです:

1. `onData` および `onDataError` は 1回または複数回実行されます。
2. 呼ばれた場合、`onError` は 1回実行されます (システムワーカーの処理を停止します)。
3. エラーが発生しなかった場合、`onResponse` が 1回実行されます。
4. `onTerminate` は常に実行されます。

戻り値

このオブジェクトに対して、SystemWorker クラスの関数やプロパティを呼び出すことができます。

Windows の例

1. 特定のドキュメントをメモ帳で開きます:

```
var $sw : 4D.SystemWorker
var $options : Object
$options:=New object
$options.hideWindow:= False

$sw:=4D.SystemWorker.new ("C:\\\\WINDOWS\\\\notepad.exe C:\\\\Docs\\\\new folder\\\\res.txt";$options)
```

2. コンソールで npm install を実行します:

```
var $folder : 4D.Folder
var $options : Object
var $worker : 4D.SystemWorker

$folder:=Folder(fk database folder)
$options:=New object
$options.currentDirectory:=$folder
$options.hideWindow:=False

$worker:=4D.SystemWorker.new("cmd /c npm install";$options)
```

3. 特定のドキュメントを Microsoft® Word® アプリケーションで開きます:

```
$mydoc:="C:\\Program Files\\Microsoft Office\\Office15\\WINWORD.EXE C:\\Tempo\\output.txt"
var $sw : 4D.SystemWorker
$sw:=4D.SystemWorker.new($mydoc)
```

4. カレントディレクトリでコマンドを実行し、メッセージを送信します:

```

var $param : Object
var $sys : 4D.SystemWorker

$param:=New object
$param.currentDirectory:=Folder(fk database folder)
$sys:=4D.SystemWorker.new("git commit -F -";$param)
$sys.postMessage("This is a postMessage")
$sys.closeInput()

```

5. ユーザーが Windows上で外部ドキュメントを開くのを許可します:

```

$docname:=Select document("");"*.*";"開くファイルを選択してください";0
If(OK=1)
    var $sw : 4D.SystemWorker
    $sw:=4D.SystemWorker.new("cmd.exe /C start \"\" \"$docname\"\"")
End if

```

macOS の例

1. テキストファイルを編集します (`cat` はファイルを編集するための macOS コマンドです)。この例題では、コマンドのフルアクセスパスを渡しています:

```

var $sw : 4D.SystemWorker
$sw:=4D.SystemWorker.new("/bin/cat /folder/myfile.txt")
$sw.wait() //同期的実行

```

2. 独立した "グラフィック" アプリケーションを起動するには、`open` システムコマンドの使用が推奨されます (これは、アプリケーションをダブルクリックするのと同じ効果を持ちます)。

```

var $sw : 4D.SystemWorker
$sw:=4D.SystemWorker.new ("open /Applications/Calculator.app")

```

3. "Users" フォルダーの中身を取得します (`ls -l` は、DOS の `dir` に相当する macOS のコマンドです)。

```

var $systemworker : 4D.SystemWorker
var $output : Text
var $errors : Collection

$systemworker:=4D.SystemWorker.new("/bin/ls -l /Users ")
$systemworker.wait(5)
$output:=$systemworker.response
$error:=$systemworker.errors

```

4. 上記と同じコマンドで、"Params" ユーザークラスを使ったコールバック関数の処理方法を示しています:

```

var $systemworker : 4D.SystemWorker
$systemworker:=4D.SystemWorker.new("/bin/ls -l /Users ";cs.Params.new())

// "Params" クラス

Class constructor
    This.dataType:="text"
    This.data:=""
    This.dataError:=""

Function onResponse($systemWorker : Object)
    This._createFile("onResponse"; $systemWorker.response)

Function onData($systemWorker : Object; $info : Object)
    This.data+=$info.data
    This._createFile("onData";this.data)

Function onDataError($systemWorker : Object; $info : Object)
    This.dataError+=$info.data
    This._createFile("onDataError";this.dataError)

Function onTerminate($systemWorker : Object)
    var $textBody : Text
    $textBody:="Response: "+$systemWorker.response
    $textBody+="ResponseError: "+$systemWorker.responseError
    This._createFile("onTerminate"; $textBody)

Function _createFile($title : Text; $textBody : Text)
    TEXT TO DOCUMENT(Get 4D folder(Current resources folder)+$title+".txt"; $textBody)

```

.closeInput()

▶ 覆歴

.closeInput() | 引数 | タイプ | | 説明 | | -- | --- | ::| ----- | | | | このコマンドは引数を必要としません |

説明

`.closeInput()` 関数は、外部プロセスの入力ストリーム (`stdin`) を閉じます。

`.closeInput()` は、`postMessage()` を介した全データの受信を待機している実行ファイルに、データ送信が終了したことを知らせるのに便利です。

例題

```

// gzip するデータを作成します
var $input;$output : Blob
var $gzip : Text
TEXT TO BLOB("Hello, World!";$input)
$gzip:="\"C:\\Program Files (x86)\\GnuWin32\\bin\\gzip.exe\" "

// 非同期のシステムワーカーを作成します
var $worker : 4D.SystemWorker
$worker:= 4D.SystemWorker.new($gzip;New object("dataType";"blob"))

// stdin に圧縮ファイルを送信します
$worker.postMessage($input)
// 終了したことを明確にするため closeInput() を呼び出します
// gzip (および stdin からのデータを待機する多数のプログラム) は入力ストリームが明示的に閉じられるまで待機します
$worker.closeInput()
$worker.wait()

$output:=$worker.response
$worker.postMessage($input)
// 終了したことを明確にするため closeInput() を呼び出します
// gzip (および stdin からのデータを待機する多数のプログラム) は入力ストリームが明示的に閉じられるまで待機します
$worker.closeInput()
$worker.wait()

$output:=$worker.response
$worker.postMessage($input)
// 終了したことを明確にするため closeInput() を呼び出します
// gzip (および stdin からのデータを待機する多数のプログラム) は入力ストリームが明示的に閉じられるまで待機します
$worker.closeInput()
$worker.wait()

$output:=$worker.response

```

.commandLine

.commandLine : Text

説明

.commandLine プロパティは、 new() 関数に引数として渡したコマンドラインを格納します。

このプロパティは 読み取り専用 です。

.currentDirectory

.currentDirectory : 4D.Folder

説明

.currentDirectory プロパティは、外部プロセスが実行される作業ディレクトリを格納します。

.dataType

.dataType : Text

説明

.dataType プロパティは、レスポンス本文のデータ型を格納します。とりうる値: "text" または "blob"。

このプロパティは 読み取り専用 です。

.encoding

.encoding : Text

説明

.encoding プロパティは、レスポンス本文のエンコーディングを格納します。このプロパティは `dataType` が "text" のときにのみ利用できます。

このプロパティは 読み取り専用 です。

.errors

.errors : Collection

説明

.errors プロパティは、実行エラーの場合、4Dエラーのコレクションを格納します。

コレクションの各要素は、以下のプロパティを持つオブジェクトです：

プロパティ	タイプ	説明
[].errorCode	number	4Dエラーコード
[].message	テキスト	4Dエラーの詳細
[].componentSignature	テキスト	エラーを返した内部コンポーネントの署名

エラーが発生しなかった場合、.errors は空のコレクションを格納します。

.exitCode

.exitCode : Integer

説明

.exitCode プロパティは、外部プロセスから返される終了コードを格納します。プロセスが正常に終了しなかった場合、exitCode は `undefined` です。

このプロパティは 読み取り専用 です。

.hideWindow

.hideWindow : Boolean

説明

.hideWindow プロパティは、実行ファイルや DOSコンソールのウィンドウを隠すのに使用できます (Windows のみ)。

読み書き可能 プロパティです。

.pid

.pid : Integer

説明

.pid プロパティは、外部プロセスの、システムレベルでの一意的な識別子を格納します。

このプロパティは 読み取り専用 です。

.postMessage()

.postMessage(*message* : Text)
.postMessage(*messageBLOB* : Blob)

引数	タイプ	説明
<i>message</i>	テキスト	-> 外部プロセスの入力ストリーム (stdin) に書き込むテキスト
<i>messageBLOB</i>	BLOB	-> 入力ストリームに書き込むバイト数

説明

.postMessage() 関数は、外部プロセスの入力ストリーム (stdin) への書き込みをおこないます。*message* には *stdin* に書き込むテキストを渡します。

.postMessage() 関数は、*stdin* に渡す BLOB型の *messageBLOB* 引数も受け取るため、バイナリデータを送信することもできます。

options オブジェクト の .dataType プロパティを使って、レスポンス本文が BLOB を返すようにできます。

.response

.response : Text
.response : Blob

説明

.response プロパティは、リクエストが終了した時点で、返された全データの結合を格納します（つまり、プロセスの出力から取得された全メッセージ）。

メッセージのデータ型は dataType 属性によって定義されています。

このプロパティは 読み取り専用 です。

.responseError

.responseError : Text

説明

.responseError プロパティは、リクエストが終了した時点で、返された全エラーの結合を格納します。

.terminate()

.terminate() | 引数 | タイプ | | 説明 | | -- | --- | ::| ----- | | | | このコマンドは引数を必要としません |

説明

.terminate() 関数は、SystemWorker の実行を強制終了します。

この関数は、システムワーカーを終了して実行中のスクリプトに制御を戻す命令を送ります。

.terminated

.terminated : Boolean

説明

`.terminated` プロパティは、外部プロセスが終了された場合に `true`を格納します。

このプロパティは 読み取り専用 です。

.timeout

`.timeout : Integer`

説明

`.timeout` プロパティは、外部プロセスが生きている場合、キルされるまでの秒数を格納します。

このプロパティは 読み取り専用 です。

.wait()

▶ 履歴

`.wait({ timeout : Real }) : 4D.SystemWorker`

引数	タイプ		説明
timeout	実数	->	待機時間 (秒単位)
戻り値	4D.SystemWorker	<-	SystemWorker オブジェクト

説明

`.wait()` 関数は、`SystemWorker` の実行終了まで、または `timeout` で指定した時間が経過するまで待機します。

`timeout` には、秒単位の値を渡します。`SystemWorker` スクリプトは、`timeout` に指定された時間だけ、外部プロセスを待ちます。`timeout` を省略した場合、スクリプトの実行は無期限に待機します。

実際には、`.wait()` はタイムアウトに達した場合を除き、`onTerminate` フォーミュラのプロセス終了まで待ちます。タイムアウトに達した場合、`SystemWorker` はキルされません。

`.wait()` の実行中、コールバック関数、とくに他のイベントや他の `SystemWorker` インスタンスからのコールバックは実行されます。コールバックから `terminate()` を呼び出すことで、`.wait()` を終了することができます。

この関数は、`SystemWorker` オブジェクトを返します。

`SystemWorker` を 4D のワーカープロセスの形で作成した場合、この関数は必要ありません。

WebServer

`WebServer` クラス API を使って、メイン（ホスト）アプリケーションおよび、各コンポーネントの Web サーバーを開始・モニターすることができます（[Web サーバーオブジェクト 参照](#)）。このクラスは `4D` クラスストアより提供されます。

Web サーバーオブジェクト

Web サーバーオブジェクトは `WEB Server` コマンドによってインスタンス化されます。

これらは、次のプロパティや関数を持ちます：

概要

<code>.accessKeyDefined : Boolean</code> Web サーバーの設定にアクセスキーが定義されていれば true
<code>.certificateFolder : Text</code>
<code>.characterSet : Number</code> <code>.characterSet : Text</code>
<code>.cipherSuite : Text</code>
<code>.CORSEnabled : Boolean</code>
<code>.CORSSettings : Collection</code>
<code>.debugLog : Number</code>
<code>.defaultHomepage : Text</code>
<code>.HSTSEnabled : Boolean</code>
<code>.HSTSMaxAge : Number</code>
<code>.HTTPCompressionLevel : Number</code>
<code>.HTTPCompressionThreshold : Number</code>
<code>.HTTPEnabled : Boolean</code>
<code>.HTTPPort : Number</code>

.HTTPTrace : Boolean
.HTTPSEnabled : Boolean
.HTTPSSPort : Number HTTPS のリッスンIPポート番号
.inactiveProcessTimeout : Number 旧式セッションプロセスの非アクティブタイムアウト時間 (分単位)
.inactiveSessionTimeout : Number 旧式セッションの非アクティブタイムアウト時間 (分単位; cookie にて設定)
.IPAddressToListen : Text
.isRunning : Boolean Webサーバーの実行状態
.keepSession : Boolean Webサーバーがスケーラブルセッションを使用している場合に true、それ以外は false
.logRecording : Number
.maxConcurrentProcesses : Number
.maxRequestSize : Number
.maxSessions : Number 旧式セッションにおける同時セッションの最大数
.minTLSVersion : Number
.name : Text Webサーバーアプリケーションの名称
.openSSLVersion : Text 使用されている OpenSSLライブラリのバージョン
.perfectForwardSecrecy : Boolean サーバーの PFS利用可否状況
.rootFolder : Text
.scalableSession : Boolean Webサーバーがスケーラブルセッションを使用している場合に true、それ以外は false

.sessionCookieDomain : Text

```

| | .sessionCookieName : Text
| | .sessionCookiePath : Text
セッションcookie の "path" フィールド| | .sessionCookieSameSite : Text
| | .sessionIPAddressValidation : Boolean
セッションcookie の IP アドレス検証| | .start() : Object
.start( settings : Object ) : Object
対象の Webサーバーを開始させます| | .stop()
対象の Webサーバーを停止します|

```

WEB Server

▶履歴

WEB Server : 4D.WebServer

WEB Server(option : Integer) : 4D.WebServer

引数	タイプ		説明
option	Integer	->	取得する Webサーバー (省略時のデフォルト = Web server database)
戻り値	4D.WebServer	<-	WebServer オブジェクト

WEB Server コマンドは、デフォルトの Webサーバーオブジェクト、または option 引数で指定された Webサーバーオブジェクトを返します。

optionが省略された場合のデフォルトでは、このコマンドはデータベースの Webサーバー (デフォルトWebサーバー) への参照を返します。取得する Webサーバーを指定するには、option に以下の定数のいずれか一つを渡してください:

定数	値	説明
Web server database	1	カレントデータベースの Webサーバー(省略時のデフォルト)
Web server host database	2	コンポーネントのホストデータベースの Webサーバー
Web server receiving request	3	リクエストを受け取った Webサーバー (ターゲットWebサーバー)

返される Webサーバーオブジェクトには、Webサーバープロパティのカレント値が格納されています。

例題

返される Webサーバーオブジェクトには、Webサーバープロパティのカレント値が格納されています。

```

// コンポーネントのメソッド
var $hostWS : 4D.WebServer
$hostWS:=WEB Server(Web server host database)
If($hostWS.isRunning)
  ...
End if

```

WEB Server list

▶履歴

WEB Server list : Collection

引数	タイプ		説明
戻り値	Collection	<-	利用可能な Webサーバーオブジェクトのコレクション

.stop() 関数は、対象の Webサーバーを停止します。

`WEB Server list` コマンドは、4Dアプリケーション内で利用可能な Webサーバーオブジェクトのコレクションを返します。

- ホストデータベースの Webサーバーを1つ（デフォルトWebサーバー）
- コンポーネント毎の Webサーバー各1つ

4Dアプリケーションは一つ以上の Webサーバーを持つことが可能です：

デフォルトの Webサーバーオブジェクトは、4D 起動時に自動的にロードされます。その一方で、コンポーネントの Webサーバーは、`WEB Server` コマンドによってそれぞれインスタンス化しなくてはなりません。

サーバーが実際に実行中か否かに関わらず、`WEB Server list` コマンドは利用可能な Webサーバーをすべて返します。

例題

Webサーバーオブジェクトの `.name` プロパティを使用することで、リスト内の各 Webサーバーオブジェクトが関連づけられているデータベースまたはコンポーネントを識別することができます。

```
var $wSList : Collection
var $vRun : Integer

$wSList:=WEB Server list
$vRun:=$wSList.countValues(True;"isRunning")
ALERT("利用可能 Webサーバー "+String($wSList.length)+" つ中、 "+String($vRun)+" つの Webサーバーが実行中です。")
```

.accessKeyDefined

`.accessKeyDefined` : Boolean

`.accessKeyDefined` プロパティは、Webサーバーの設定にアクセスキーが定義されていれば `true`を格納します。このプロパティは `WebAdmin` Webサーバーによって、管理インターフェースのセキュリティ設定を有効化するのに使用されます。

.certificateFolder

`.certificateFolder` : Text

認証ファイルが保存されているフォルダーのパス。パスは、ファイルシステムを使用した POSIXフルパスの形式です。`.start()` 関数に渡す `settings` 引数内でこのプロパティを使用する場合、`Folder` オブジェクトも使用可能です。

.characterSet

`.characterSet` : Number
`.characterSet` : Text

アプリケーションに接続してくるブラウザーとの通信に 4D Webサーバーが使用すべき文字セット。デフォルト値は OS の言語に依存します。値には、MIBenum 整数や名称の文字列、IANA が定義する識別子を使用できます。以下は、4D Webサーバーがサポートしている文字セットに対応する識別子のリストです：

- 4 = ISO-8859-1
- 12 = ISO-8859-9
- 13 = ISO-8859-10
- 17 = Shift-JIS
- 2024 = Windows-31J
- 2026 = Big5
- 38 = euc-kr
- 106 = UTF-8
- 2250 = Windows-1250

- 2251 = Windows-1251
- 2253 = Windows-1253
- 2255 = Windows-1255
- 2256 = Windows-1256

.cipherSuite

.cipherSuite : Text

保護されたプロトコルのために使用される暗号スイートリスト。これは、4D Webサーバーが実装する暗号化アルゴリズムの優先順位を設定します。セミコロン区切りの文字列として設定できます (例: "ECDHE-RSA-AES128-...")。詳細は Open SSL サイトの [ciphers ページ](#) を参照ください。

.CORSEnabled

.CORSEnabled : Boolean

Web サーバーの CORS (*Cross-origin resource sharing*、オリジン間リソース共有) サービス状態。セキュリティ上の理由により、"ドメイン間" のリクエストはブラウザーレベルでデフォルトで禁止されています。有効化されている場合 (true)、ドメイン外 Web ページからの XHR コール (REST リクエストなど) をアプリケーションにおいて許可することができます (CORS ドメインリストに許可されたアドレスのリストを定義する必要があります。後述の [CORSSettings 参照](#))。無効化されている場合 (false、デフォルト) には、CORS で送信されたサイト間リクエストはすべて無視されます。有効時 (true) に、許可されていないドメインやメソッドがサイト間リクエストを送信した場合、"403 - forbidden" エラーレスポンスによって拒否されます。

デフォルト: false (無効)

デフォルト: false (無効)

.CORSSettings

.CORSSettings : Collection

CORS サービスに許可されたホストとメソッドの一覧 ([CORSEnabled プロパティ参照](#))。各オブジェクトは必ず host プロパティを格納していかなければなりません。methods プロパティは任意です。

- host (テキスト、必須): CORS を介したサーバーへのデータリクエスト送信が許可されている外部ページのドメイン名または IP アドレス。複数のドメインを追加してホワイトリストを作成することができます。host が存在しない、または空の場合、当該オブジェクトは無視されます。複数のシングルクォーテーションがサポートされています:
 - 192.168.5.17:8081
 - 192.168.5.17
 - 192.168.*
 - 192.168.*:8081
 - <http://192.168.5.17:8081>
 - http://*.myDomain.com
 - <http://myProject.myDomain.com>
 - *.myDomain.com
 - myProject.myDomain.com
 - *
- methods (テキスト、任意): 対応する CORS ホストに対して許可する HTTP メソッド。メソッド名はセミコロン区切りで指定します (例: "post;get")。methods が空、null、あるいは undefined の場合、すべてのメソッドが許可されます。

.debugLog

.debugLog : Number

HTTP リクエストログファイルの状態 (アプリケーションの "Logs" フォルダーに格納されている HTTPDebugLog_nn.txt ファイル (nn はファイル番号))。

- 0 = 無効
- 1 = 有効、リクエスト本文なし (本文サイズあり)

- 3 = 有効、レスポンスの本文のみ
- 5 = 有効、リクエストの本文のみ
- 7 = 有効、リクエストおよびレスポンスの本文あり

.defaultHomepage

.defaultHomepage : Text

デフォルトのホームページの名称 または、カスタムのホームページを送信しない場合は ""。

.HSTSEnabled

.HSTSEnabled : Boolean

HTTP Strict Transport Security (HSTS) 状態。HSTS によって、Webサーバーはブラウザーに対し、セキュアな HTTPS接続のみを許可する宣言できます。Webサーバーからの初回レスポンスを受け取った際にブラウザーは HSTS情報を記録し、以降の HTTPリクエストは自動的に HTTPSリクエストに変換されます。ブラウザー側でこの情報が保存される時間は `HSTSMaxAge` プロパティによって指定されます。HSTS のためには、サーバー上で HTTPS が有効になっていなければなりません。また、初回のクライアント接続を許可するために、HTTP も有効でなければなりません。

.HSTSMaxAge

.HSTSMaxAge : Number

新規クライアント接続ごとに HSTS がアクティブな最長時間 (秒単位)。この情報はクライアント側で指定された時間のあいだ保存されます。

デフォルト値: 63072000 (2年)。

.HTTPCompressionLevel

.HTTPCompressionLevel : Number

4D HTTPサーバーの HTTP圧縮通信 (クライアントリクエストまたはサーバーレスpons) における圧縮レベル。このセレクターを使って、実行速度を優先するか (圧縮少)、それとも圧縮レベルを優先するか (速度減) を指定し、通信を最適化することができます。

とりうる値:

- 1 から 9 (1 が低圧縮、9 が高圧縮)。
- -1 = 圧縮速度と圧縮率の妥協点を設定する

とりうる値:

.HTTPCompressionThreshold

.HTTPCompressionThreshold : Number

HTTP圧縮のしきい値 (バイト単位)。このサイズ未満のリクエストについては、通信が圧縮されません。この設定は、通信サイズが小さい場合、圧縮に処理時間が費やされるのを避けるのに有用です。

デフォルトのしきい値 = 1024 バイト

.HTTPEnabled

.HTTPEnabled : Boolean

HTTPSプロトコル状態。

.HTTPPort

.HTTPPort : Number

HTTPプロトコルの状態。

HTTP のリッスンIPポート番号。

.HTTPTrace

.HTTPTrace : Boolean

HTTP TRACE の有効化状態。セキュリティ上の理由により、Webサーバーはデフォルトで HTTP TRACE リクエストをエラー405 で拒否します。有効化されている場合、HTTP TRACE リクエストに対して Webサーバーは、リクエスト行、ヘッダー、および本文を返します。

.HTTPSEnabled

.HTTPSEnabled : Boolean `.start()` 関数は、任意の *settings* オブジェクト引数に設定したプロパティを使用して、対象の Webサーバーを開始させます。

.HTTPSPort

.HTTPSPort : Number HTTPS のリッスンIPポート番号。

HTTPS のリッスンIPポート番号。

.inactiveProcessTimeout

.inactiveProcessTimeout : Number

スケーラブルセッションモード の場合には、このプロパティは返されません。

旧式セッションプロセスの非アクティブタイムアウト時間 (分単位)。すると、On Web Legacy Close Session データベースメソッドが呼び出され、旧式セッションのコンテキストは削除されます。

デフォルト = 480 分

.inactiveSessionTimeout

.inactiveSessionTimeout : Number

スケーラブルセッションモード の場合には、このプロパティは返されません。

旧式セッションの非アクティブタイムアウト時間 (分単位; cookie にて設定)。タイムアウト時間が経過するとセッションcookie が無効になり、HTTPクライアントによって送信されなくなります。

デフォルト = 480 分

.IPAddressToListen

.IPAddressToListen : Text

4D Webサーバーが HTTPリクエストを受信する IPアドレス。デフォルトでは、特定のアドレスは定義されていません。IPv6 および IPv4 文字列形式の両方がサポートされています。

.isRunning

.isRunning : Boolean

読み取り専用プロパティ。

Webサーバーで旧式セッションが有効されている場合に true、それ以外は false。

.keepSession

.keepSession : Boolean

Webサーバーがスケーラブルセッションを使用している場合に true、それ以外は false。

参照:

[.scalableSession](#)

.logRecording

.logRecording : Number

リクエストログ (logweb.txt) の記録オプション値。

- 0 = 記録しない (デフォルト)
- 1 = CLF形式で記録する
- 2 = DLF形式で記録する
- 3 = ELF形式で記録する
- 4 = WLF形式で記録する

.maxConcurrentProcesses

.maxConcurrentProcesses : Number

Webサーバーにてサポートする最大同時Webプロセス数。この数値 (マイナス1) に達すると、4D はプロセスを作成しなくなり、新規リクエストに対して HTTPステータス 503 - Service Unavailable を返します。

とりうる値: 10 - 32000

デフォルト = 100

.maxRequestSize

.maxRequestSize : Number

Webサーバーが処理してよい HTTPリクエスト (POST) の最大サイズ (バイト単位)。最大値 (2147483647) に設定した場合、実際には制限無しということになります。制限を設けることで、サイズが非常に大きいリクエストによって Webサーバーが過負荷状態に陥ることを防ぎます。リクエストのサイズが制限に達していると、Webサーバーによって拒否されます。

とりうる値: 500000 - 2147483647

.maxSessions

.maxSessions : Number

スケーラブルセッションモード の場合には、このプロパティは返されません。

旧式セッションにおける同時セッションの最大数。制限に達すると、Webサーバーが新規セッションを作成するときに、一番古い旧式セッションが閉じられます (On Web Legacy Close Session データベースメソッドが呼び出されます)。旧式セッションの同時セッション数は、Webプロセスの合計値を超えることはできません (maxConcurrentProcesses プロパティ、デフォルト値は 100)。

.minTLSVersion

.minTLSVersion : Number

接続に必要な最低TLSバージョン。これよりも低いバージョンのみをサポートするクライアントからの接続は拒否されます。

とりうる値:

- 1 = TLSv1_0
- 2 = TLSv1_1
- 3 = TLSv1_2 (デフォルト)
- 4 = TLSv1_3

変更した場合、設定を反映するには Webサーバーを再起動する必要があります。

.name

.name : Text

読み取り専用プロパティ。

Webサーバーアプリケーションの名称。

.openSSLVersion

.openSSLVersion : Text

読み取り専用プロパティ。

使用されている OpenSSLライブラリのバージョン。

.perfectForwardSecrecy

.perfectForwardSecrecy : Boolean

読み取り専用プロパティ。

サーバーの PFS利用可否状況。

.rootFolder

.rootFolder : Text

Webサーバーのルートフォルダーのパス。パスは、ファイルシステムを使用した POSIXフルパスの形式です。 `settings` 引数内でこのプロパティを使用する場合、Folder オブジェクトも使用可能です。

.scalableSession

.scalableSession : Boolean

Webサーバーがスケーラブルセッションを使用している場合に true、それ以外は false。

参照:

[.keepSession](#)

.sessionCookieDomain

.sessionCookieDomain : Text

セッションcookie の "path" フィールド。セッションcookie のスコープを制御するのに使用されます。たとえば、このセレクターに "/4DACTION" という値を設定した場合、4DACTION で始まる動的リクエストの場合にのみクライアントは cookie を送信し、ピクチャーや静的ページへのリクエストは除外されます。

.sessionCookieName

.sessionCookieName : Text

セッションID の保存に使用されるセッションcookie の名称。

読み取り専用プロパティ。

.sessionCookiePath

.sessionCookiePath : Text

セッションcookie の "path" フィールド。セッションcookie のスコープを制御するのに使用されます。たとえば、このセレクターに "/*.4d.fr" の値を設定した場合、リクエストの宛先が ".4d.fr" のドメインに限り、クライアントは cookie を送信します。

.sessionCookieSameSite

▶ 履歴

.sessionCookieSameSite : Text

セッションcookie の "SameSite" 属性の値。どうる値 (定数使用):

定数	値	説明
Web SameSite Strict	"Strict"	デフォルト値 - ファーストパーティのコンテキストでのみ cookie が送信されます。
Web SameSite Lax	"Lax"	サイト間のサブリクエストにおいても cookie が送信されますが、ユーザーがリンクを辿って大元のサイトに戻る場合に限ります。
Web SameSite None	"None"	ファーストパーティーやオリジン間リクエストにかわらず、すべてのコンテキストにおいて cookie が送信されます。

[Web Server オブジェクト](#) の設定は、読み取り専用プロパティ

([.isRunning](#)、[.name](#)、[.openSSLVersion](#)、[.perfectForwardSecrecy](#)、[.sessionCookieName](#)) を除いて、すべてカスタマイズ可能です。

.sessionIPAddressValidation

.sessionIPAddressValidation : Boolean

スケーラブルセッションモード の場合には、このプロパティは使用されません (IPアドレスを検証しません)。

セッションcookie の IP アドレス検証。セキュリティ上の理由により、セッションcookie を持つ各リクエストに対して Webサーバーはデフォルトで IPアドレスを検証します。アプリケーションによっては、この検証機能を無効化し、IPアドレスが合致しなくてもセッションcookie を受け入れるようにしたいかもしれません。たとえば、モバイルデバイスが WiFi と 3G/4G ネットワークを切り替えた場合、IPアドレスが変更されます。このように IPアドレスが変更しても、クライアントによる Webセッションの継続を許可できます (アプリケーションのセキュリティレベルは下がります)。

.start()

▶ 履歴

.start() : Object

.start(*settings* : Object) : Object

引数	タイプ		説明
settings	Object	->	開始時の Webサーバー設定
戻り値	Object	<-	Webサーバー開始のステータス

*読み取り専用プロパティ。 * Webサーバーの実行状態。

プロジェクトの設定ファイルに定義されているデフォルトの設定、または `WEB SET OPTION` コマンドで定義された設定（ホストデータベースのみ）を使用して、Webサーバーは開始されます。しかし、`settings` 引数を渡せば、Webサーバーセッションにおいてカスタマイズされた設定を定義することができます。

`Web Server オブジェクト` の設定は、読み取り専用プロパティ (`.isRunning`、`.name`、`.openSSLVersion`、`.perfectForwardSecrecy`、`.sessionCookieName`) を除いて、すべてカスタマイズ可能です。

データベースWebサーバーを停止します：

返されるオブジェクト

関数は Webサーバーの開始ステータスを表すオブジェクトを返します。このオブジェクトには、次のプロパティが格納があります：

プロパティ		タイプ	説明
success		Boolean	Webサーバーが正常に開始された場合には <code>true</code> 、それ以外は <code>false</code>
errors		Collection	エラースタック (Webサーバーが正常に開始された場合には返されません)
	<code>[] .errCode</code>	Number	4Dエラーコード
	<code>[] .message</code>	Text	4Dエラーの詳細
	<code>[] .componentSignature</code>	Text	エラーを返した内部コンポーネントの署名

Webサーバーが既に起動していた場合、エラーが返されます。

例題

```
var $settings;$result : Object
var $webServer : 4D.WebServer

$settings:=New object("HTTPPort";8080;"defaultHomepage";"myAdminHomepage.html")

$webServer:=WEB Server
$result:=$webServer.start($settings)
If($result.success)
// ...
End if
```

.stop()

▶ 補足

`.stop()`

| 引数 | タイプ | 説明 | | -- | --- | | ----- | | | | このコマンドは引数を必要としません |

カスタマイズされた設定は `.stop()` が呼び出されたときにリセットされます。

Webサーバーが開始されている場合は、処理中のリクエストが完了次第、すべての Web接続と Webプロセスが閉じられます。Webサーバーが開始されていなかった場合、関数はなにもしません。

この関数は、`.start()` 関数の `settings` 引数を使用してセッションに対して定義したカスタマイズされた Web設定があった場合、それらをリセットします。

例題

データベースWebサーバーを停止します：

```
var $webServer : 4D.WebServer  
  
$webServer:=WEB Server(Web server database)  
$webServer.stop()
```

ZIPArchive

4D ZIP アーカイブは、一つ以上のファイルまたはフォルダーを格納している `File` または `Folder` オブジェクトで、元のサイズより小さくなるように圧縮されているものをいいます。これらのアーカイブは ".zip" 拡張子を持つように作成され、ディスクスペースの確保や、サイズ制限があるメディア（例：メールまたはネットワークなど）経由のファイル転送を容易にする用途に使用できます。

- 4D ZIPアーカイブは [ZIP Create archive](#) コマンドで作成します。
- 4D `ZIPFile` および `ZIFolder` インスタンスは、[ZIP Read archive](#) コマンドによって返されるオブジェクトの `root` プロパティ (`ZIFolder`) を通して利用可能です。

例題

`ZIPFile` オブジェクトを取得し、その中身を確認します：

```
var $path; $archive : 4D.File
var $zipFile : 4D.ZipFile
var $zipFolder : 4D.ZipFolder
var $txt : Text

$path:=Folder(fk desktop folder).file("MyDocs/Archive.zip")
$archive:=ZIP Read archive($path)
$zipFolder:=$archive.root // 圧縮ファイルのルートフォルダーを保存します
$zipFile:=$zipFolder.files()[0] // 最初のファイルを読み取ります

If($zipFile.extension=".txt")
    $txt:=$zipFile.getText()
End if
```

概要

`.root : 4D.ZipFolder`
ZIPアーカイブのコンテンツにアクセスするためのバーチャルフォルダー

ZIP Create archive

▶ 補足

`ZIP Create archive (fileToZip : 4D.File ; destinationFile : 4D.File) : Object`

`ZIP Create archive (folderToZip : 4D.Folder ; destinationFile : 4D.File { ; options : Integer }) : Object`

`ZIP Create archive (zipStructure : Object ; destinationFile : 4D.File) : Object`

引数	タイプ		説明
<code>fileToZip</code>	<code>4D.File</code>	->	圧縮する File または Folder オブジェクト
<code>folderToZip</code>	<code>4D.Folder</code>	->	圧縮する File または Folder オブジェクト
<code>zipStructure</code>	<code>Object</code>	->	圧縮する File または Folder オブジェクト
<code>destinationFile</code>	<code>4D.File</code>	->	アーカイブの保存先ファイル
<code>options</code>	<code>Integer</code>	->	<code>folderToZip</code> オプション: <code>ZIP Without enclosing folder</code> (外側のフォルダーを除外して ZIP圧縮をおこなう)
戻り値	<code>Object</code>	<-	ステータスオブジェクト

説明

`ZIP Create archive` コマンドは、圧縮された ZIPArchive オブジェクトを作成し、その処理のステータスを返します。

第1引数として、4D.File、4D.Folder、あるいは zipStructure オブジェクトを渡すことができます。

- `fileToZip`: 圧縮する `4D.File` オブジェクトを引数として渡します。
- `folderToZip`: 圧縮する `4D.Folder` を渡します。この場合、任意の `options` 引数を渡して、フォルダーのコンテンツのみを圧縮（つまり、外側のフォルダを除外）することができます。 `ZIP Create archive` はデフォルトで、フォルダーとその中身を圧縮するので、展開処理をしたときにはフォルダーを再作成します。フォルダーの中身のみを解凍処理で復元するには、`options` 引数に `ZIP Without enclosing folder` 定数を渡します。
- `zipStructure`: ZIPArchive オブジェクトを表すオブジェクトを引数として渡します。以下のプロパティを利用して、このオブジェクトを定義することができます:

プロパティ	タイプ	説明												
compression	整数	<ul style="list-style-type: none"> • <code>ZIP Compression standard</code> : Deflate圧縮（デフォルト） • <code>ZIP Compression LZMA</code> : LZMA圧縮 • <code>ZIP Compression XZ</code> : XZ圧縮 • <code>ZIP Compression none</code> : 圧縮なし 												
level	Integer	<p>圧縮レベル。とりうる値: 1 - 10。低い値ではファイルが大きくなり、高い値ではファイルが小さくなります。ただし、圧縮レベルはパフォーマンスに影響します。デフォルト値（省略時）:</p> <ul style="list-style-type: none"> • <code>ZIP Compression standard</code> : 6 • <code>ZIP Compression LZMA</code> : 4 • <code>ZIP Compression XZ</code> : 4 												
encryption	整数	<p>パスワードが設定されていた場合に使用する暗号化方法:</p> <ul style="list-style-type: none"> • <code>ZIP Encryption AES128</code> : 128-bit キーを使った AES による暗号化 • <code>ZIP Encryption AES192</code> : 192-bit キーを使った AES による暗号化 • <code>ZIP Encryption AES256</code> : 256-bit キーを使った AES による暗号化（パスワードが設定されている場合のデフォルト） • <code>ZIP Encryption none</code> : 暗号化なし（パスワードが設定されてない場合のデフォルト） 												
password	Text	暗号化が必要な場合に使用するパスワード												
files	Collection	<ul style="list-style-type: none"> • <code>4D.File</code> または <code>4D.Folder</code> オブジェクトのコレクション • 以下のプロパティを持ったオブジェクトのコレクション: <table border="1"> <thead> <tr> <th>プロパティ</th> <th>タイプ</th> <th>説明</th> </tr> </thead> <tbody> <tr> <td>source</td> <td>4D.File または 4D.Folder</td> <td>File または Folder</td> </tr> <tr> <td>destination</td> <td>テキスト</td> <td>（任意） - アーカイブのコンテンツ構成を変更するための相対ファイルパス</td> </tr> <tr> <td>option</td> <td>number</td> <td>（任意） - <code>ZIP Ignore invisible files</code> で非表示ファイルを無視、0 を渡すと全ファイルを圧縮</td> </tr> </tbody> </table> 	プロパティ	タイプ	説明	source	4D.File または 4D.Folder	File または Folder	destination	テキスト	（任意） - アーカイブのコンテンツ構成を変更するための相対ファイルパス	option	number	（任意） - <code>ZIP Ignore invisible files</code> で非表示ファイルを無視、0 を渡すと全ファイルを圧縮
プロパティ	タイプ	説明												
source	4D.File または 4D.Folder	File または Folder												
destination	テキスト	（任意） - アーカイブのコンテンツ構成を変更するための相対ファイルパス												
option	number	（任意） - <code>ZIP Ignore invisible files</code> で非表示ファイルを無視、0 を渡すと全ファイルを圧縮												
callback	4D.Function	\$1 に圧縮の進捗 (0 - 100) を受け取るコールバックフォーミュラ												

`destinationFile` には、作成する ZIPアーカイブ（名前や位置など）を記述する `4D.File` オブジェクトを渡します。作成した ZIPアーカイブがあらゆるソフトウェアで自動的に処理されるようにするために、".zip" 拡張子の使用が推奨されます。

アーカイブが作成されると、[ZIP Read archive](#) を使用してアクセスすることができます。

ステータスオブジェクト

戻り値のステータスオブジェクトには、以下のプロパティが格納されています:

プロパティ	タイプ	説明
statusText	Text	エラーメッセージ (あれば): <ul style="list-style-type: none"> • ZIPアーカイブを開けません • ZIPアーカイブを作成できません • 暗号化にはパスワードが必要です
status	Integer	ステータスコード
success	Boolean	アーカイブが正常に作成された場合には true、それ以外は false

例題 1

4D.File を圧縮します:

```
var $file; $destination : 4D.File
var $status : Object

$destination:=Folder(fk desktop folder).file("MyDocs/file.zip")
$file:=Folder(fk desktop folder).file("MyDocs/text.txt")

$status:=ZIP Create archive($file;$destination)
```

例題 2

フォルダー自体は圧縮せずに 4D.Folder の中身だけを圧縮します:

```
var $folder : 4D.Folder
var $destination : 4D.File
var $status : Object

$destination:=Folder(fk desktop folder).file("MyDocs/Images.zip")
$folder:=Folder(fk desktop folder).folder("MyDocs/Images")

$status:=ZIP Create archive($folder;$destination;ZIP Without enclosing folder)
```

例題 3

ZIPアーカイブの圧縮にパスワードと進捗バーを使います:

```
var $destination : 4D.File
var $zip;$status : Object
var progID : Integer

$destination:=Folder(fk desktop folder).file("MyDocs/Archive.zip")

$zip:=New object
$zip.files:=Folder(fk desktop folder).folder("MyDocs/Resources").folders()
$zip.password:="password"
$zip.callback:=Formula(myFormulaCompressingMethod($1))

progID:=Progress New // 4D Progress コンポーネントを使います

$status:=ZIP Create archive($zip;$destination)

Progress QUIT(progID)
```

myFormulaCompressingMethod :

```
var $1 : Integer  
Progress SET PROGRESS(progID;Num($1/100))
```

例題 4

`zipStructure` オブジェクトに、圧縮したいフォルダーとファイルを格納したコレクションを渡します:

```
var $destination : 4D.File  
var $zip;$err : Object  
$zip:=New object  
$zip.files:=New collection  
$zip.files.push(New object("source";Folder(fk desktop folder).file("Tests/text.txt")))  
$zip.files.push(New object("source";Folder(fk desktop folder).file("Tests/text2.txt")))  
$zip.files.push(New object("source";Folder(fk desktop folder).file("Images/image.png")))  
  
$destination:=Folder(fk desktop folder).file("file.zip")  
$err:=ZIP Create archive($zip;$destination)
```

例題 5

高い圧縮レベルの代替圧縮アルゴリズムを使用します:

```
var $destination : 4D.File  
var $zip; $err : Object  
  
$zip:=New object  
$zip.files:=New collection  
$zip.files.push(Folder(fk desktop folder).folder("images"))  
$zip.compression:=ZIP Compression LZMA  
$zip.level:=7 // デフォルト値は 4 です  
  
$destination:=Folder(fk desktop folder).file("images.zip")  
$err:=ZIP Create archive($zip; $destination)
```

ZIP Read archive

▶ 覆歴

ZIP Read archive (`zipFile` : 4D.File { ; `password` : Text }) : 4D.ZipArchive

引数	タイプ		説明
<code>zipFile</code>	4D.File	->	ZIPアーカイブファイル
<code>password</code>	Text	->	ZIPアーカイブのパスワード (必要であれば)
戻り値	4D.ZipArchive	<-	アーカイブオブジェクト

説明

`ZIP Read archive` コマンドは、`zipFile` のコンテンツを取得し、`4D.ZipArchive` オブジェクト形式で返します。

このコマンドは ZIPアーカイブを展開することはしません。その中身に関する情報を提供するのみです。アーカイブのコンテンツを取り出すには、`file.copyTo()` あるいは `folder.copyTo()` などの関数を使用します。

`zipFile` 引数として、圧縮された ZIPアーカイブを参照している `4D.File` オブジェクトを渡します。ターゲットのアーカイブファイルは `ZIP Read archive` が実行を終えるまで (全コンテンツ/参照が取得/解放されるまで) は開いた状態となり、その後自動的に閉じられます。

`zipFile` 引数で指定した ZIPファイルがパスワードで保護されていた場合、任意の `password` 引数を渡してパスワードを提供する必要があります。パ

スワードが必要にも関わらず、コンテンツ読み出し時にパスワードが提示されなかった場合、エラーが生成されます。

アーカイブオブジェクト

戻り値の `4D.ZipArchive` オブジェクトは単一の `root` プロパティを格納しており、その値は `4D.ZipFolder` オブジェクトです。このフォルダーは ZIPアーカイブの全コンテンツを表します。

例題

ZIPFile オブジェクトを取得し、その中身を確認します：

```
var $archive : 4D.ZipArchive
var $path : 4D.File

$path:=Folder(fk desktop folder).file("MyDocs/Archive.zip")
$archive:=ZIP Read archive($path)
```

アーカイブ内のファイルとフォルダーの一覧を取得します：

```
$folders:=$archive.root.folders()
$files:=$archive.root.files()
```

ファイルのコンテンツを、root フォルダーから取り出すことなく読み出します：

```
If($files[$i].extension=".txt")
    $txt:=$files[$i].getText()
Else
    $blob:=$files[$i].getContent()
End if
```

root フォルダーから取り出します：

```
// 特定のファイルを取得します
$folderResult:=$files[$i].copyTo(Folder(fk desktop folder).folder("MyDocs"))

// すべてのファイルを取得します
$folderResult:=$archive.root.copyTo(Folder(fk desktop folder).folder("MyDocs"))
```

.root

`.root` : 4D.ZipFolder

説明

`.root` プロパティは、ZIPアーカイブのコンテンツにアクセスするためのバーチャルフォルダーを格納します。

`root` フォルダーとそのコンテンツは、[ZipFile](#) および [ZipFolder](#) の関数とプロパティを使用することで操作可能です。

このプロパティは 読み取り専用 です。

ZIPFile

[File](#) クラスの次のプロパティや関数は `ZIPFile` オブジェクトにおいて利用可能です:

ZIPFile で利用可能な File API	説明
<code>.copyTo(destinationFolder : 4D.Folder { ; newName : Text } { ; overwrite : Integer }) : 4D.File</code>	
<code>.creationDate : Date</code>	
<code>.creationTime : Time</code>	
<code>.exists : Boolean</code>	
<code>.extension : Text</code>	
<code>.fullName : Text</code>	
<code>.getContent() : 4D.Blob</code>	
<code>.getIcon({ size : Integer }) : Picture</code>	
<code>.getText({ charSetName : Text { ; breakMode : Integer } }) : Text</code> <code>.getText({ charSetNum : Integer { ; breakMode : Integer } }) : Text</code>	
<code>.hidden : Boolean</code>	
<code>.isAlias : Boolean</code>	
<code>.isFile : Boolean</code>	
<code>.isFolder : Boolean</code>	
<code>.isWritable : Boolean</code>	ZIPアーカイブの場合は常に <code>false</code>
<code>.modificationDate : Date</code>	
<code>.modificationTime : Time</code>	
<code>.name : Text</code>	
<code>.original : 4D.File</code> <code>.original : 4D.Folder</code>	
<code>.parent : 4D.Folder</code>	
<code>.path : Text</code>	アーカイブを起点とした相対パスを返します
<code>.platformPath : Text</code>	

ZIPFolder

[Folder](#) クラスの次のプロパティや関数は `ZIPFolder` オブジェクトにおいて利用可能です:

ZIPFolder で利用可能な <code>Folder</code> API	説明
<code>.copyTo(destinationFolder : 4D.Folder { ; newName : Text } { ; overwrite : Integer }) : 4D Folder</code>	
<code>.creationDate : Date</code>	アーカイブ内のフォルダーと <code>root</code> フォルダーの日付が異なる場合があります
<code>.creationTime : Time</code>	アーカイブ内のフォルダーと <code>root</code> フォルダーの時刻が異なる場合があります
<code>.exists : Boolean</code>	
<code>.extension : Text</code>	
<code>.file(path : Text) : 4D.File</code>	
<code>.files({ options : Integer }) : Collection</code>	
<code>.folder(path : Text) : 4D.Folder</code>	
<code>.folders({ options : Integer }) : Collection</code>	
<code>.fullName : Text</code>	
<code>.getIcon({ size : Integer }) : Picture</code>	
<code>.hidden : Boolean</code>	
<code>.isAlias : Boolean</code>	
<code>.isFile : Boolean</code>	
<code>.isFolder : Boolean</code>	
<code>.isPackage : Boolean</code>	
<code>.modificationDate : Date</code>	アーカイブ内のフォルダーと <code>root</code> フォルダーの日付が異なる場合があります
<code>.modificationTime : Time</code>	アーカイブ内のフォルダーと <code>root</code> フォルダーの時刻が異なる場合があります
<code>.name : Text</code>	
<code>.original : 4D.Folder</code>	
<code>.parent : 4D.Folder</code>	アーカイブの仮想 <code>root</code> フォルダーは親を持ちません。しかしながら、アーカイブ内のフォルダーは <code>root</code> 以外の親を持つ場合があります。
<code>.path : Text</code>	アーカイブを起点とした相対パスを返します
<code>.platformPath : Text</code>	

ナビゲーションドロップダウン

ナビゲーションドロップダウンは、コードを整理し、クラスやメソッド内の移動を助けるツールです。

The screenshot shows a code editor interface with a navigation dropdown menu open. The menu contains the following items:

- set fullName
- constructor (highlighted with a blue background)
- get fullName
- set fullName
- Fix following lines
- Functions
- doSomething

The code in the editor is:

```
1  v Class constructor($firstname : Text; $lastname : Text)
2      This.firstname := $firstname
3      This.lastname := $lastname
4
5  v Function get fullName () -> $fullName : Text
6      $fullName := This.firstname + " " + This.lastname
7
```

いくつかのタグは自動的に追加されますが、マークを使ってドロップダウンリストを補完することもできます。

コードのナビゲーション

ドロップダウンリストの項目をクリックすると、当該項目のコードの先頭行に移動します。また、矢印キーで項目を選択し Enter キーで決定して移動することもできます。

自動タグ

コンストラクター、メソッド宣言、関数、計算属性は自動的にタグ付けされ、ドロップダウンリストに追加されます。

クラス/メソッドにタグがない場合、ツールは "タグなし" アイコンを表示します。

次の項目が自動的に追加されます:

アイコン	項目
Ø	タグなし
🎯	クラスコンストラクターまたはメソッド宣言
⌚	計算属性 (get, set, orderBy, query)
f	クラス関数名

手動タグ

コードにマークを追加して、以下のタグをドロップダウンに追加できます:

アイコン	項目
🔖	MARK: タグ
📝	TODO: タグ
🔧	FIXME: タグ

タグは、次のようにコメントを付けて宣言します:

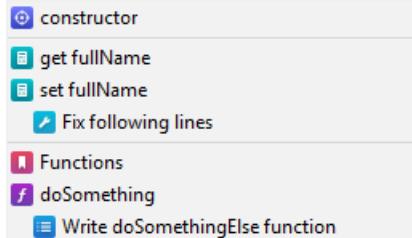
```
// FIXME: 要修正
```

タグ宣言の際、文字の大小は区別されません。 `fixme:` と記述することもできます。

MARK: タグの後にハイフンを付けると、コードエディターとドロップダウンメニューに区切り線が引かれます。つまり、次のように書くと:

```
// FIXME: Fix following lines  
  
This.firstName:=Substring($fullName; 1; $p-1)  
This.lastName:=Substring($fullName; $p+1)  
  
//MARK:- Functions  
  
Function doSomething  
  
// TODO: Write doSomethingElse function
```

このような結果になります:



関数内のマーカーはドロップダウンリスト内でインデント（字下げ）されますが、関数末尾の MARK: タグの後に指示がない場合を除きます。この場合は、インデントなしで表示されます。

表示順

タグは、メソッド/クラス内の出現順に表示されます。

メソッドやクラスのタグをアルファベット順に表示するには、次のいずれかを実行します:

- ドロップダウンツールを 右クリックする
- macOS では option、Windows では Alt を押しながら、ドロップダウンツールをクリックします。

関数内のタグは、親項目と一緒に移動します。

画面の説明

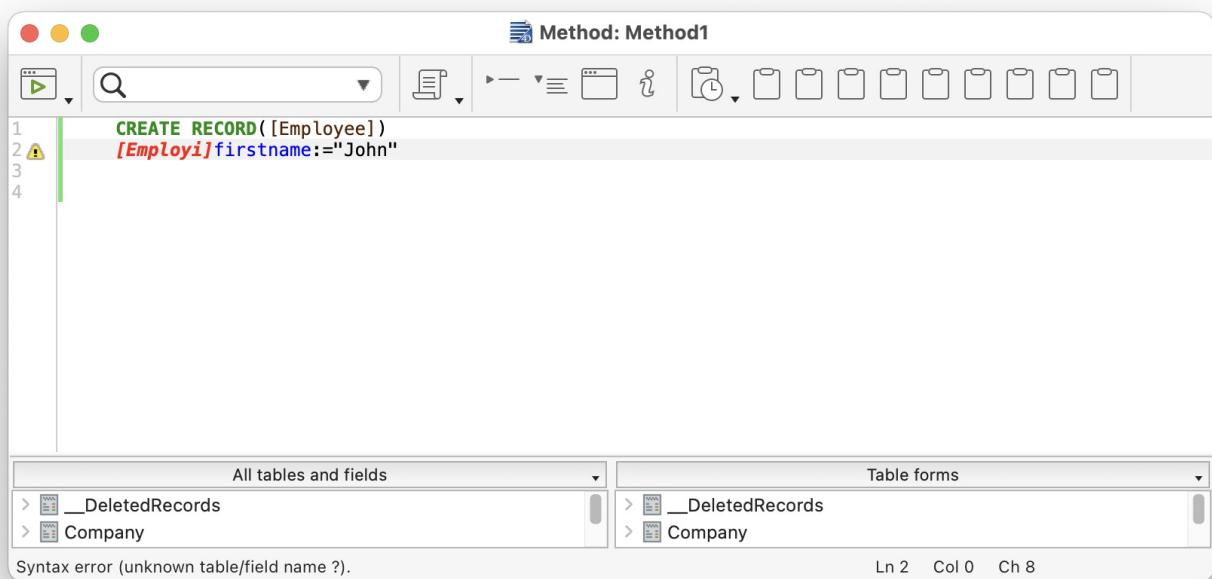
エラーは日常的なものです。相当行数のコードを書いているのに、1つもエラーが出ないというのは非常にまれです。むしろ、エラーに対応・修正することは普通のことなのです。

4D の開発環境には、あらゆる種類のエラーに対応するためのデバッグツールが用意されています。

エラーの種類

タイプミス

タイプミスはメソッドエディターによって検出されます。これらは赤色で示され、ウィンドウ下部に追加情報が表示されます。以下はタイプミスの例です：



The screenshot shows the 4D Method Editor window titled "Method: Method1". The code area contains the following lines:

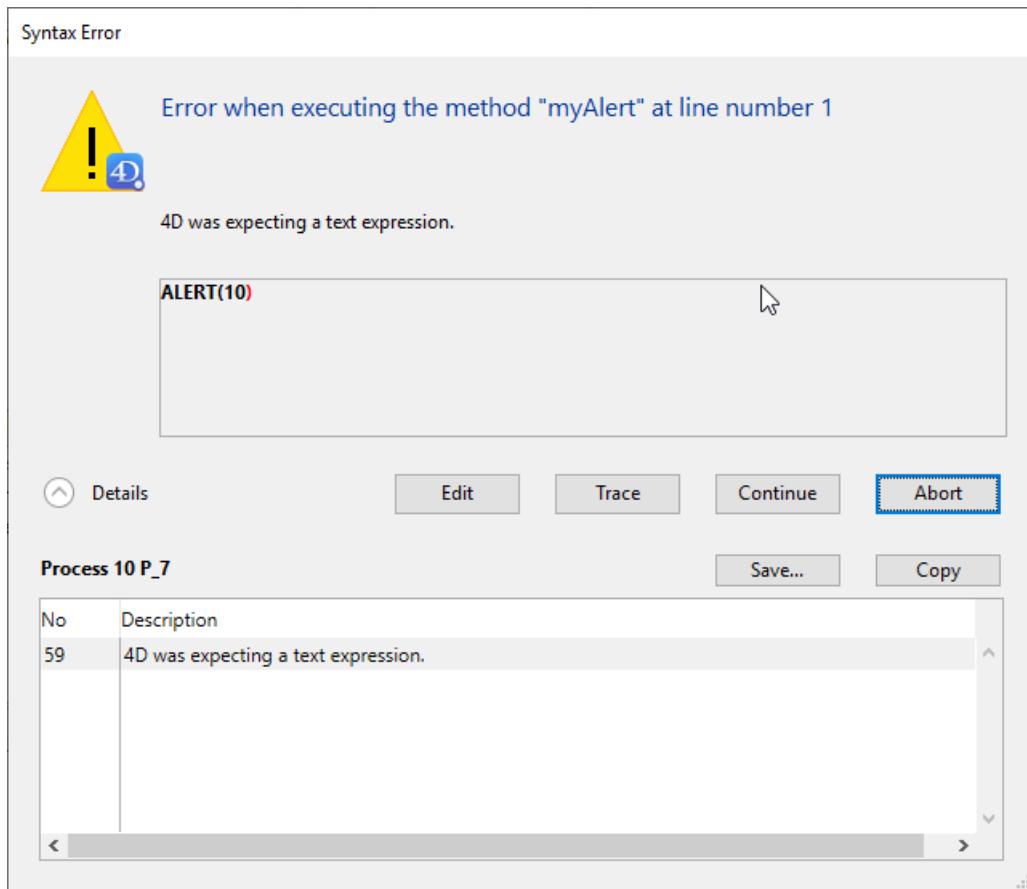
```
1 CREATE RECORD([Employee])
2 [Employi]firstname := "John"
3
4
```

Line 2 has a yellow warning icon next to the opening brace of the record definition. The code editor interface includes a toolbar at the top with various icons, and two panels at the bottom: "All tables and fields" and "Table forms", both listing tables like "__DeletedRecords" and "Company". A status bar at the bottom right indicates "Ln 2 Col 0 Ch 8".

このようなタイプミスは通常、シンタックスエラーの原因となります（上の例では、テーブル名が間違っています）。コードの該当行の編集を確定すると、エラーの説明が表示されます。このような場合タイプミスを修正して Enterキーを押すと、再度コードの検証がおこなわれます。

シンタックスエラー

メソッドの実行時に限って、とらえることのできるエラーがあります。 [シンタックスエラーウィンドウ](#) はエラーが発生した際に表示されます：たとえば：



詳細 エリアを展開すると、最新のエラーと番号が表示されます。

環境エラー

時に、BLOB を作成するための十分なメモリがない場合があります。ディスク上のドキュメントにアクセスしようとした時にドキュメントが存在しないか、他のアプリケーションにより既に開かれていることもあります。このようなエラーは、コードやその書き方を直接の原因として発生するわけではありません。ほとんどの場合、このようなエラーは `ON ERR CALL` コマンドでインストールされた [エラー処理メソッド](#) で簡単に対処できます。

設計またはロジックエラー

一般に、これらは発見が最も難しいタイプのエラーです。これまでに説明しているエラーは、タイプミスを除いて、"設計またはロジックのエラー" という範疇に該当します。これらを検知するには、[デバッガー](#) を使用します。たとえば:

- まだ初期化されていない変数を用いようとしたため、シンタックスエラー が発生する場合があります。
- 間違った引数を受け取ったサブルーチンが、その間違った名前によりドキュメントを開こうとしたため、環境エラー が発生している場合があります。

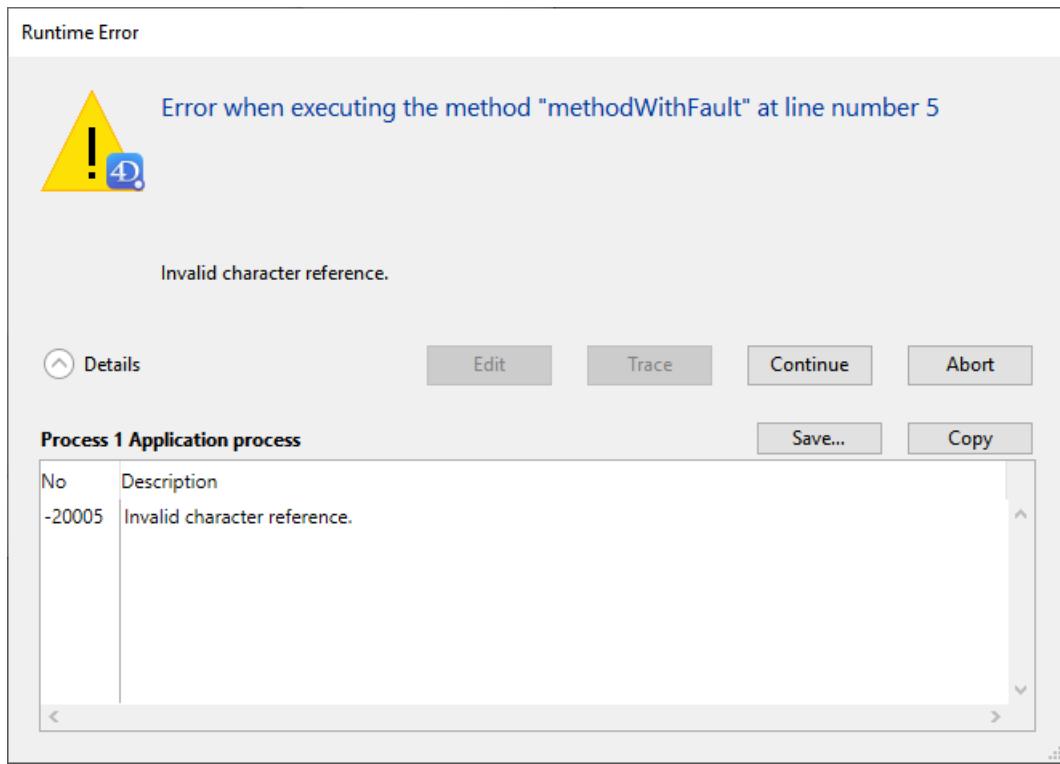
設計またはロジックのエラーには、次のような場合もあります:

- `SAVE RECORD` コマンドを呼び出す際に、対象となるレコードがロックされているかどうかを最初にテストしなかったために、レコードが正しく更新されない。
- オプション引数を追加した状態がテストされていないため、メソッドが想定通りに動作しない。

場合によって問題の原因是、実際に中断が発生しているコード部分ではなく、外部にあることもあります。

ランタイムエラー

アプリケーションモードでは、インターフリターモードでは決して見られないエラーが発生する場合があります。次に例を示します:

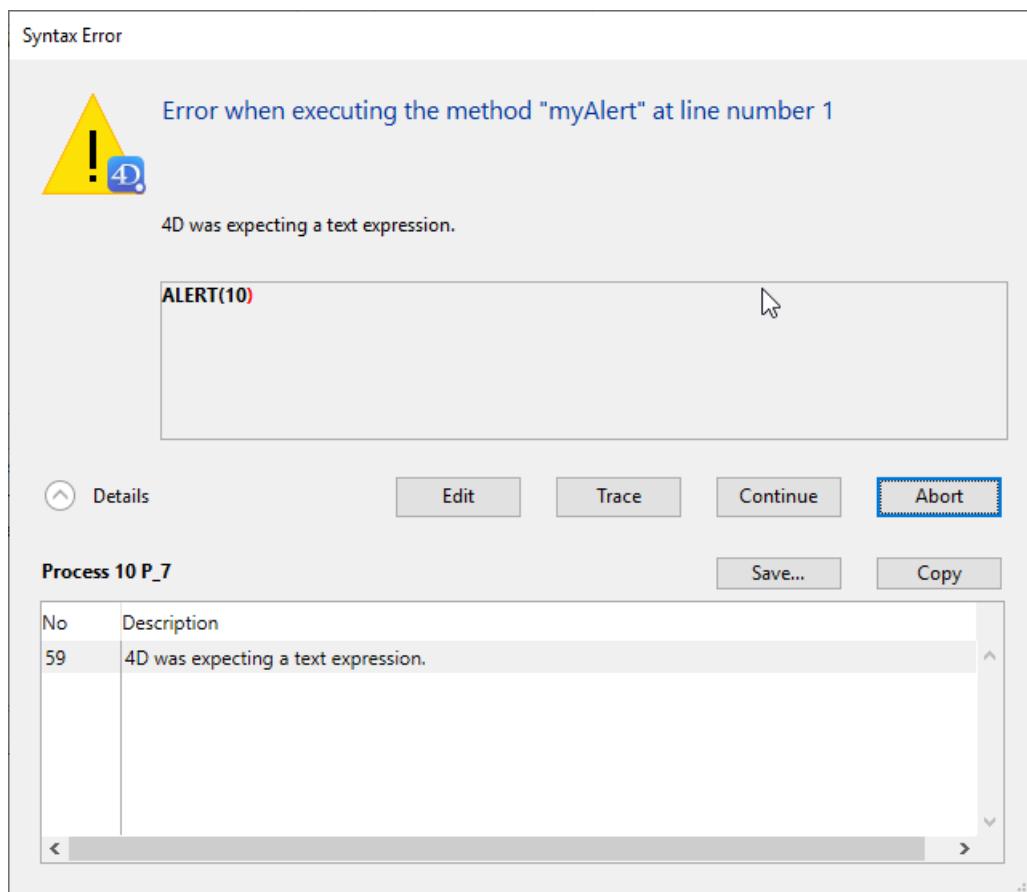


問題の原因を迅速に発見するには、メソッドの名前と行番号を記録し、ストラクチャーファイルのインタープリター版を再び開いて、メソッドの該当行を確認します。

シンタックスエラーウィンドウ

メソッドの実行が停止されるとシンタックスエラーウィンドウが表示されます。これは以下の理由で起ります:

- 以降のメソッド実行を妨げるエラーが発生した。
- メソッドが False の表明を生成した (`ASSERT` コマンド参照)。



上部テキストエリアには、エラーの説明メッセージが表示されます。下部テキストエリアには、エラーが発生した時の実行行が表示されます。エラーが発生

したエリアはハイライトされます。詳細ボタンをクリックすると、プロセスのエラースタックを表示するエリアを展開できます。

シンタックスエラーウィンドウにはいくつかのオプションが用意されています:

- 編集: すべてのメソッド実行が中断されます。4D はデザインモードに切り替わり、エラーが発生したメソッドがメソッドエディターで表示され、エラーを修正することができます。原因に心当たりがあり、これ以上調査しなくても修正できる場合にこのオプションを使用します。
- トレース: トレース/デバッガーモードに入ります。デバッガーウィンドウが表示されます。該当行の一部が未実行の場合には、トレースボタンを数回クリックする必要があるかもしれません。
- 続行: 実行が継続されます。エラーが発生した行は、エラーの位置によっては一部のみ実行済みである場合があります。慎重に実行を継続してください: エラーが原因で、メソッドの残り部分が正常に実行できない場合があります。SET WINDOW TITLE のように、コードの残りの部分の実行やテストの妨げにならない単純な呼び出しでエラーが発生している場合にのみ、続行ボタンをクリックすることを推奨します。

Tips: ループ中などで繰り返し発生するエラーの場合には、続行ボタンを無視ボタンに変更できます。続行ボタンが最初に現れたときに、Altキー(Windows)またはOptionキー(macOS)を押しながらボタンをクリックします。すると、同じエラーによってダイアログが呼び出されたときには、ボタンラベルが無視へと変化します。

- アボート: メソッドが中断され、メソッドの実行を開始する前の状態に戻ります:
 - イベントに対してフォームメソッドまたはオブジェクトメソッドが実行されている場合には、これらは停止され、フォームに戻ります。
 - メソッドがアプリケーションモードから実行されている場合には、このモードに戻ります。
- コピー: デバッグ情報をクリップボードにコピーします。この情報はエラーの内部環境(番号や内部コンポーネント等)を説明します。情報はタブ区切り形式で記述されます。
- 保存...: シンタックスエラーウィンドウの内容とコールチェーンを.txtファイルに保存します。

デバッガー

エラー検出の際によくある初歩的な失敗は、シンタックスエラーウィンドウのアボートボタンをクリックし、メソッドエディターに戻り、コードを表示して原因を確認しようとすることです。これは止めてください。デバッガーを常に使用すれば、相当の時間と労力を節減することができます。

デバッガーを使うと、メソッドをステップごとにゆっくりと実行することができます。デバッガーは、エラーが発生した理由を知るために必要な情報を表示できます。この情報があれば、エラーの修正方法はわかります。

デバッガーを使用するもう1つの理由は、コードの作成です。いつも以上に複雑なアルゴリズムを作成してしまう場合があります。達成感こそありますが、コーディングが正しいかどうかは100%確かとはいえません。見当もつかないまま実行するのではなく、コードの最初で TRACE コマンドを使用します。その後、コードをステップごとに実行して、動作を監視することができます。

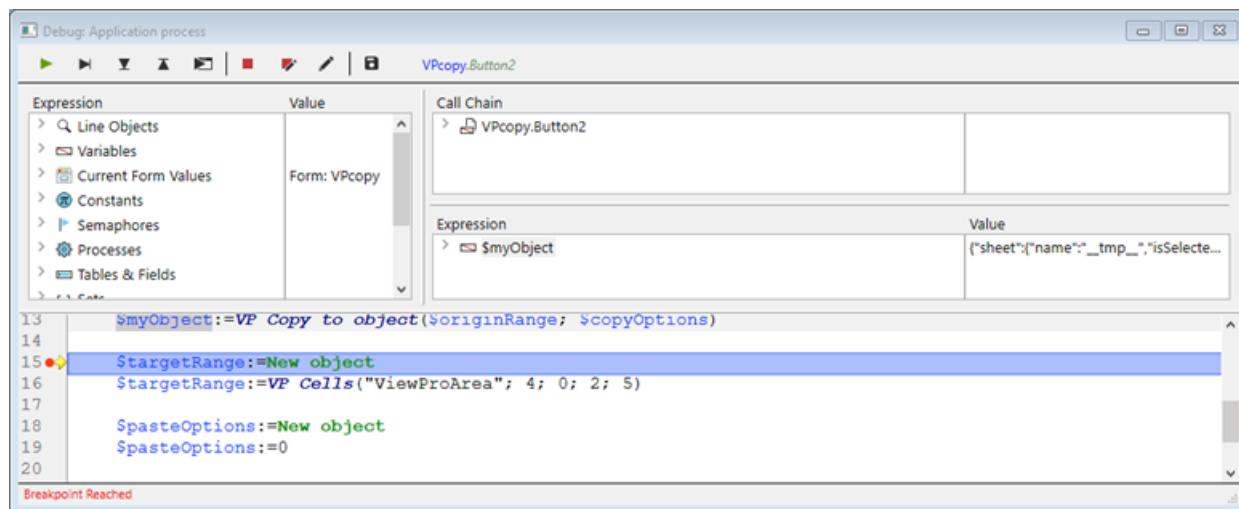
ブレーク

デバッグ作業では、コードの一部のトレースを特定の行までスキップする必要がある場合があります。また、ある式が特定の値になった時(例: "\$myVar > 1000")や、特定の4Dコマンドが呼び出されるたびにコードをトレースしたい場合もあります。

このようなニーズに対応するために、ブレークポイントとキヤッショマンド機能が用意されています。これらの機能は、メソッドエディター、デバッガー、ランタイムエクスプローラーから設定できます。

デバッガー

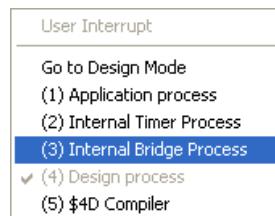
デバッガーは、エラーを発見したり、メソッドの実行を監視する必要がある場合に便利です。デバッガーを使って、コードをステップごとにゆっくり確認して情報を検証することができます。このようにメソッドをステップごとに確認する処理はトレースと呼ばれます。



デバッガーの呼び出し

デバッガーを開くには、次のような方法があります：

- シングルクエラーウィンドウで トレース ボタンをクリックする。
- TRACE コマンドを使用する。
- メソッド実行ウィンドウで デバッグ ボタンをクリックする、またはメソッドエディターで 実行してデバッグ ボタンを選択する。
- メソッド実行中に Alt+Shift+右クリック (Windows) または Ctrl+Option+Cmd+クリック (Macintosh) をおこない、表示されるポップアップ ウィンドウ内でトレースするプロセスを選択する：



- ランタイムエクスプローラーのプロセスページにてプロセスを選択した後、トレース ボタンをクリックする。
- メソッドエディターウィンドウ、またはランタイムエクスプローラーのブレークおよびキャッチページでブレークポイントを作成する。

デバッガーウィンドウは、現在トレースしているメソッドまたはクラス関数の名前や、デバッガーが表示される原因となったアクションの情報を表示します。上のウィンドウの例では、次の情報が表示されています：

- 現在トレースされているメソッドは *Clients_BuildLogo* メソッドです。
- デバッガーウィンドウが表示されているのは、キャッチコマンドの対象に設定された C_PICTURE コマンドへの呼び出しが検出されたためです。

新しいデバッガーウィンドウの表示には、同じセッション内で表示された最後のデバッガーウィンドウと同じ構成（ウィンドウのサイズと位置、分割線の配置および式評価エリアの内容）を使用します。複数のユーザー プロセスを実行した場合には、それぞれのプロセスを個別にトレースできます。つまり、各プロセスにつき 1 つのデバッガーウィンドウを表示できます。

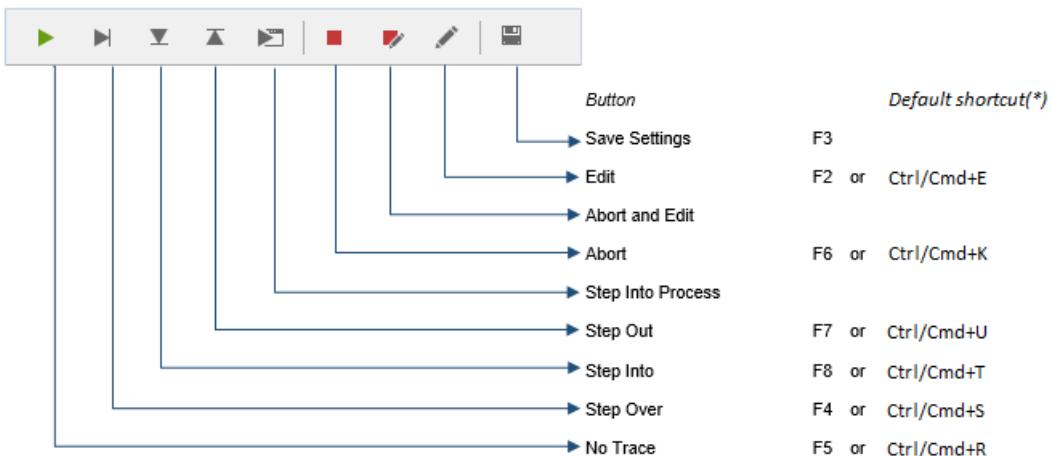
デバッガーウィンドウは、一般的にそのコードが実行されているマシン上に表示されます。シングルユーザー版アプリケーションの場合、デバッガーは常にアプリケーションを実行しているマシン上に表示されます。クライアント/サーバー版アプリケーションの場合は：

- ローカルで実行されているコードの場合には、リモート 4D 上に表示されます。
- サーバー上で実行されているコード（サーバー上で実行 オプションがつけられたメソッド）の場合には、サーバーマシン上に表示されます。

ヘッダレスモードで実行中のサーバーでは、デバッガーウィンドウを表示することはできません。この場合はリモートデバッガーを使用する必要があります。[リモートマシンからのデバッグ](#) 参照。

ツールバー ボタン

デバッガーウィンドウの上部にある実行制御ツールバーには、デフォルトショートカットが設定された複数のボタンがあります：



デフォルトのショートカットは、環境設定ダイアログボックスのショートカットページで変更できます。

トレース終了

トレースが停止され、通常のメソッド実行が再開されます。

Shift + F5 または Shift を押しながら トレース終了 ボタンをクリックすると、実行が再開されます。この操作により、以降のカレントプロセスでの全ての TRACE 呼び出しが無効になります。

次行に進む

現在のメソッド行（プログラムカウンターと呼ばれる黄色い矢印で示されている行）が実行されます。その後、デバッガは次の行に移動します。

"次の行に進む" ボタンは、サブルーチンや関数に移動することではなく、現在トレースの対象となっているメソッドのレベルにとどまります。呼び出されるサブルーチンや関数もトレースしたい場合には、呼び出しメソッドもトレース ボタンを使用します。

リモートデバッグにおいて、メソッドがサーバー上で実行されていた場合には、メソッドの最後の行の実行後にその親メソッドが呼ばれます。その時、親メソッドがリモート側で実行されていた場合には、このボタンは トレース終了 ボタンと同じように振る舞います。

呼び出しメソッドもトレース

別のメソッド（サブルーチンまたは関数）を呼び出す行が実行される時にこのボタンを使用すると、呼び出されたメソッドがデバッガーウィンドウに表示され、ステップ実行できます。

デバッガーウィンドウの [呼び出し連鎖エリア](#) では、新しく呼び出されたメソッドがカレント（一番上）となります。

別のメソッドを呼び出していない行が実行される場合には、このボタンは 次行に進む ボタンと同じように振る舞います。

中断

メソッドは中断され、メソッドの実行を開始する前の状態に戻ります。

- イベントに対して実行しているフォームメソッドまたはオブジェクトメソッドをトレースしている場合には、いずれの場合にも停止され、フォームに戻ります。
- アプリケーションモードから実行しているメソッドをトレースしていた場合には、停止後そのモードに戻ります。

中断 & 編集

メソッドは中断されます。メソッドエディターウィンドウが開いて、中断 & 編集 ボタンがクリックされた時点で実行していたメソッドを表示します。

Tip: このボタンは、コードにどのような変更が必要かが明らかであり、メソッドのテストを続行するためにその変更が必要な場合に使用してください

い。変更が完了したら、メソッドを再実行できます。

編集

メソッドは一時停止されます。メソッドエディターウィンドウが開いて、編集ボタンがクリックされた時点で実行していたメソッドを表示します。

このボタンをクリックしてメソッドを編集した場合には、現在の実行は中断されないため、編集内容の反映は次回実行時になります。

Tip: このボタンは、コードに必要な変更内容がわかっている場合で、その変更がコードの残り部分の実行やトレースの妨げにならない場合に使用します。

設定保存

現在のデバッガウィンドウの構成を、デフォルト構成として保存します。構成には次の内容が含まれます:

- ウィンドウのサイズと位置
- 分割線の配置および式評価エリアの内容

これらは、プロジェクト内に保存されます。

このアクションはリモートデバッグモードでは利用できません ([リモートマシンからのデバッグ](#) 参照)。

ウォッチャエリア

ウォッチャエリア は実行コントロールツールバーの下、デバッガウィンドウの左上隅に表示されます。次に例を示します:

Expression	Value
▶ 🔎 Line Objects	
◀ 📄 Variables	
▶ 📄 Interprocess	
◀ 📄 Process	
▶ 📄 Document	""
▶ 📄 Error	0
▶ 📄 FldDelimit	9
▶ 📄 OK	0
▶ 📄 RecDelimit	13
▶ 📄 Local	
▶ 📄 Parameters	
▶ 📄 Self	Nil
▶ 📄 Current Form Values	
▶ ⚙ Constants	
▶ 🐾 Semaphores	
▶ 📄 Processes	
▶ 📄 Tables & Fields	
▶ 📄 Sets	
▶ 📄 Named Selections	
▶ 📄 Information	
▶ 📄 Web	

このエリアはリモートデバッグモードでは使用できません。

ウォッチャエリア には、システム、4D環境、および実行環境について役立つ一般情報が表示されます。

式 欄には、要素や式の名前が表示されます。 値 欄には、要素や式に対応する現在の値が表示されます。 エリア右側の値をクリックすると、その値が変更可能な場合には、要素の値を修正できます。

テーマ、テーマサブリスト (あれば)、テーマ項目は、いつでも [カスタムウォッチャエリア](#) にドラッグ & ドロップすることができます。

式リスト

ラインオブジェクト

このテーマには、次のような要素や式の値が表示されます:

- 実行されるコードの行 (プログラムカウンターにより、ソースコードエリア内で黄色の矢印でマークされている行) で使用されている。
- コードの前の行で使用されている。

コードの前の行とは実行直後の行であるため、ラインオブジェクトテーマでは、その行が実行される前または後の現在の行の要素や式が表示されます。たとえば、次のメソッドを実行した場合を想定します:

```
TRACE
$a:=1
$b:=$a+1
$c:=$a+$b
```

- ソースコードエリアのプログラムカウンターが `$a:=1` の行にセットされた状態で、デバッグウィンドウが開きます。この時点では ラインオブジェクトテーマには、次のように表示されています:

<code>\$a</code>	未定義

まだ初期化されていない変数 `$a` が表示されているのは、実行の対象となっている行で使用されているためです。

- 次行に進む ボタンをクリックします。プログラムカウンターは `$b:=$a+1` の行に設定されます。この時点では、ラインオブジェクトテーマに次のように表示されます:

<code>\$a</code>	1
<code>\$b</code>	未定義

変数 `$a` の値は 1 になりました。まだ初期化されていない変数 `$b` が表示されているのは、実行の対象となっている行で使用されているためです。

- 次行に進む ボタンをクリックします。プログラムカウンターは `$c:=$a+$b` の行に設定されます。この時点では、ラインオブジェクトテーマに次のように表示されます:

<code>\$c</code>	未定義
<code>\$a</code>	1
<code>\$b</code>	2

変数 `$b` の値が 2 になりました。まだ初期化されていない変数 `$c` が表示されているのは、実行の対象となっている行で使用されているためです。

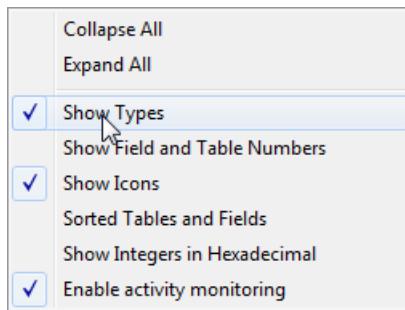
変数

このテーマは、次のサブテーマから構成されます:

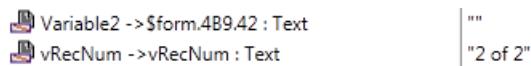
サブテーマ	説明	値は編集可能?
インタープロセス	この時点で使用されているインタープロセス変数のリスト	<input type="radio"/>
プロセス	カレントプロセスで使用されているプロセス変数のリスト	<input type="radio"/>
ローカル	現在トレースしているメソッドで使用されているローカル変数のリスト	<input type="radio"/>
パラメーター	メソッドが受け取った引数のリスト	<input type="radio"/>
Self	オブジェクトメソッドをトレースしている場合には、現在のオブジェクトへのポインター	<input checked="" type="radio"/>

他の変数と同様に、配列はそのスコープによって、インターパロセス、プロセス、およびローカルサブテーマに表示されます。デバッガーは要素ゼロと最初の100要素を表示します。値 欄で配列要素の値を変更することは可能ですが、配列のサイズを修正することはできません。

変数の型や内部名を表示するには、右クリックしてコンテキストメニューを開き、型を表示 にチェックを入れます:



このようになります:



カレントフォーム値

このテーマには、カレントフォームに含まれる各動的オブジェクトの名前に加えて、そこに関連付けられている値が表示されます:

Current Form Values		Form: debugger
bCancel		0
bDelete		0
Button3		1
bValidate		0
FirstName		"Tony"
ID		"2"
List Box1		0 elements
List Box1		Listbox sub objects

リストボックス配列などの一部のオブジェクトは、二つの異なる項目として表示されることがあります（オブジェクト自身の変数と、そのデータソース）。

定数

エクスプローラーウィンドウの定数ページのように、4D が提供する定義済み定数を表示します。このテーマの式を修正することはできません。

セマフォー

現在設定されているローカルセマフォーのリストを表示します。各セマフォーの値欄には、自身を設定したプロセスの名前が表示されます。このテーマの式を修正することはできません。グローバルセマフォーは表示されません。

プロセス

作業セッションを開始してから起動されたプロセスのリストを表示します。値欄には、各プロセスの現在の状態（実行中、一時停止等）および使用した時間が表示されます。このテーマの式を修正することはできません。

テーブルとフィールド

4Dデータベースのテーブルやフィールドのリストを表示します。各テーブル項目について、カレントプロセスにおけるカレントセクションのサイズは勿論、ロックされたレコード のナンバーも値欄に表示されます。

各フィールド項目については、カレントレコードのフィールドの値（ピクチャーと BLOB は除く）が値欄に表示されます。フィールドの値を修正することはできますが、テーブル情報を修正することはできません。

セット

カレント（トレース中の）プロセスで定義されているセットとインタープロセスセットのリストを表示します。各セットについて、レコード数とテーブル名が値欄に表示されます。このテーマの式を修正することはできません。

命名セレクション

カレント（トレース中の）プロセスで定義されている命名セレクションとインターパロセス命名セレクションのリストを表示します。各命名セレクションについて、レコード数とテーブル名が値欄に表示されます。このテーマの式を修正することはできません。

情報

このテーマは、データベースのオペレーションに関する一般的な情報を表示します。カレントのデフォルトテーブル（あれば）、物理メモリ、仮想メモリ、空きメモリ、使用中メモリ、クエリ格納先、などです。

Web

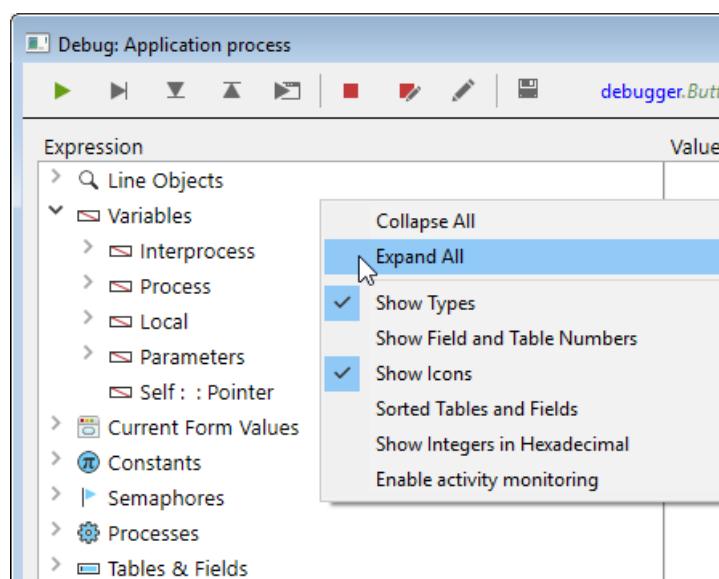
このテーマは、アプリケーションの メイン Web サーバーに関する情報が表示されます（Web サーバーが起動している場合のみ）：

- 送信する Web ファイル： 送信待機中の Web ファイルの名前（あれば）
- Web キャッシュ 利用： Web キャッシュ 内のページ数と、使用率
- Web サーバー起動時間： Web サーバーの起動時間（"時間:分:秒" 形式）
- Web ヒット回数： Web サーバー起動以降に受信した HTTP リクエストの総数と、1 秒毎の受信数
- 動作中の Web プロセス数： アクティブな Web プロセスの数と、全 Web プロセスの数

このテーマの式を修正することはできません。

コンテキストメニュー

ウォッチャーエリアのコンテキストメニューでは、追加オプションが提供されています。



- すべて閉じる： ウォッチャーエリアの階層リストの全レベルを縮小します。
- すべて拡げる： ウォッチャーエリアの階層リストの全レベルを展開します。
- 型を表示： 各項目のデータ型を（適切な場合に）表示します。
- フィールド/テーブル番号を表示： テーブルおよびフィールドの番号を表示します。テーブル番号やフィールド番号を用いて作業している場合、または `Table` や `Field` コマンドを使用し、ポインターを用いて作業している場合、このオプションは非常に便利です。
- アイコンを表示： 各項目のタイプを示すアイコンを表示します。表示速度を速くするために、このオプションをオフにすることもできます。
- テーブル/フィールドをソート： テーブルおよびフィールドをそれぞれアルファベット順に並べ替えます。
- 数値を16進で表示： 通常、数値は10進法で表示されます。このオプションを使用すると、数値が16進法表記で表示されます。注：数値を16進法で入力するには、0x（ゼロの後にx）とタイプし、その後に16進数を続けます。
- アクティビティモニターを有効にする： 動作のモニタリング（アプリケーション内部の詳細チェック）を有効にし、追加テーマ（スケジューラー、ネットワーク）に情報を表示します。

呼び出し連鎖エリア

1つのメソッドから他のメソッドまたはクラス関数が呼び出される場合があります。このエリアは、この呼び出し連鎖のリストを表示します。

Call Chain		
` thirdMethod	\$0	Undefined
` secondMethod	\$0	Undefined
\$1		->[Employee]
\$2		->[Employee]ID
\$3		Z
` firstMethod		

それぞれのメインレベルの項目は、メソッドまたはクラス関数の名前です。最も上にある項目は、現在トレース中のメソッド、次の項目は呼び出し元（トレース中メソッドを呼び出したメソッドまたはクラス関数）、その次の項目は呼び出し元の呼び出し元、のように続きます。

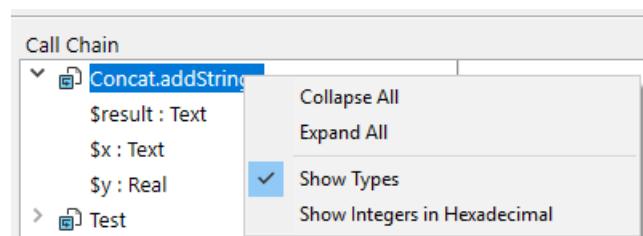
上図の例では：

- thirdMethod は引数を受け取っていません。
- \$0 は現在未定義です。これは、メソッドが \$0 に値を割り当てていないためです（メソッドがこの割り当てをまだ実行していないか、メソッドが関数ではなくサブルーチンなことが原因です）。
- secondMethod は firstMethod から 3つの引数を受け取っています：
 - \$1 は [Employee] テーブルへのポインター
 - \$2 は [Employee] テーブルの ID フィールドへのポインター
 - \$3 は値が "Z" の英数字の引数です。

呼び出し連鎖エリアのメソッド名をダブルクリックすると、そのソースコードが [ソースコードエリア](#) に表示されます。

メソッドまたは関数名の隣にあるアイコンをクリックすると、引数および戻り値のリストが展開または縮小されます。値はエリアの右側に表示されます。右の値をクリックすると、引数や戻り値の値を変更することができます。

コンテキストメニュー内の 型を表示 を選択することで、引数のデータ型を表示することができます：



メソッドの引数リストが展開されていれば、引数や戻り値を [カスタムウォッチャエリア](#) にドラッグ & ドロップすることができます。

呼び出しチェーンは [Get call chain](#) コマンドを使って取得することもできます。

カスタムウォッチャエリア

カスタムウォッチャエリアは、式を評価するために使用します。 [ウォッチャエリア](#) 似ていますが、ここでは任意の式を表示することができます。どのようなタイプの式でも評価できます：

- フィールド
- 変数
- ポインター
- 演算
- 4Dコマンド
- メソッド
- ほか値を返すものなら何でも

Expression	Value
` \$text	"Hello, World!"
` \$calcResult	3
` \$pField	->[Employee]ID
` \$myBlob	10 Ko

テキスト形式で表示できる式であれば、どのような式でも評価することができます。ピクチャーや BLOBフィールドおよび変数は表示できません。BLOB の内容を表示するには、[BLOB to text](#) のような BLOBコマンドを使用してください。

新しい式の挿入

リストに式を追加する方法は複数あります：

- ウォッчエリアまたは呼び出し連鎖エリアから項目や式をドラッグ & ドロップします。
- ソースコードエリア で式を選択し、ctrl+D (Windows) または cmd+D (macOS) を押します。
- カスタムウォッчエリアの空スペースのどこかをダブルクリックします (プレースホルダー名を持つ編集可能な式が追加されます)。

値を返すフォーミュラであれば、なんでも追加できます。

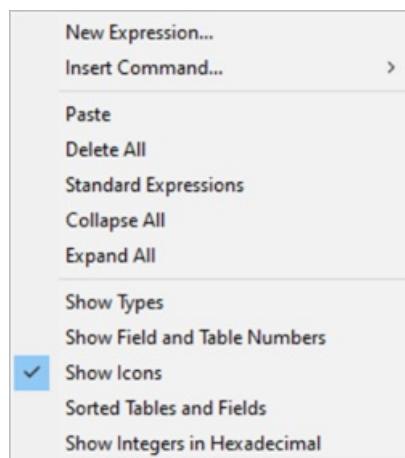
式を編集するには、その式をクリックして選択し、再びクリックすると (またはEnterキーを押す) 編集モードになります。

式を削除するには、その式をクリックして選択し、Backspace または Deleteキーを押します。

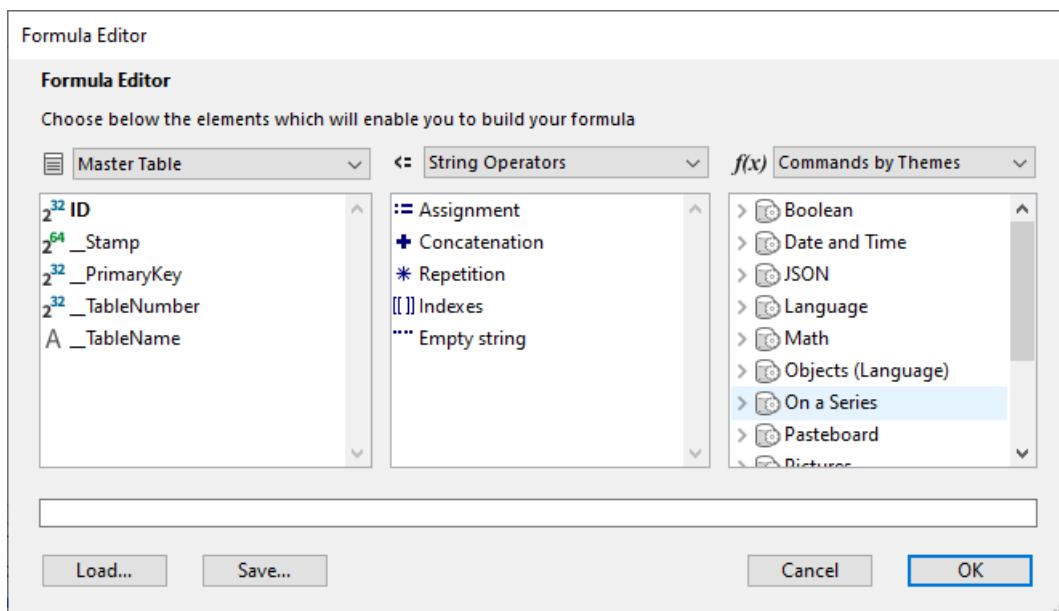
警告: システム変数 (たとえば OK変数) の値を変更するような 4D式を評価する場合、その後のメソッド実行に影響することに注意してください。

コンテキストメニュー

カスタムウォッчエリアのコンテキストメニューを使って、4D のフォーミュラエディターにアクセスできます：



新しい式...: 新しい式を挿入し、4D のフォーミュラエディターを表示します。



フォーミュラエディターに関する詳細は、[4D Design Reference マニュアル](#) を参照してください。

- コマンド挿入...: 4Dコマンドを新しい式として挿入するためのショートカット。

- すべて削除: 現在カスタムウォッチャエリアに表示されている式をすべて削除します。
- 標準式: ウォッチャエリアの式リストをコピーします。

このオプション(はリモートデバッグモードでは利用できません ([リモートマシンからのデバッグ](#) 参照))。

- すべて閉じる/すべて拡げる: 階層リストの全レベルを縮小/展開します。
- 型を表示: リストの各項目のデータ型を(適切な場合)表示します。
- フィールド/テーブル番号を表示: テーブルおよびフィールドの番号を表示します。テーブル番号やフィールド番号を用いて作業している場合、または Table や Field コマンドを使用し、ポインターを用いて作業している場合、このオプションは非常に便利です。
- アイコンを表示: 各項目のタイプを示すアイコンを表示します。
- テーブル/フィールドをソート: テーブルおよびフィールドをそれぞれアルファベット順に並べ替えます。
- 数値を16進で表示: 数値が16進法表記で表示します。数値を16進法で入力するには、0x(ゼロの後にx)とタイプし、その後に16進数を続けます。

ソースコードエリア

ソースコードエリアには、トレース中のメソッドや関数のソースコードが表示されます。

このエリアでは、[ブレークポイント](#) の追加や削除も可能です。

Tips

式の上にマウスカーソルを移動すると、Tipsとして次の内容が表示されます:

- 宣言された式の型
- 式のカレント値

```

1      // $1 contains the primary key of the manager
2      // $2 is a pointer to the resulting array
3
4  QUERY([Employee];[Employee]ID;=$1) // finds the employee whose primary key w
5  APPEND TO ARRAY($2->:[Employee]firstname+" "+[Employee]lastname) // and adds
6          [$2:Pointer = ->$reportingEmps (Get All Reporting Emps)]
7  QUERY([Employee];[Employee]managerID;=$1:[Employee]ID) // now finds all direct
8
9  If (Records in selection([Employee])>0) // if there are some
10
11    C_LONGINT($i)
12    ARRAY LONGINT($IDS;0)
13    SELECTION TO ARRAY([Employee]ID;$IDS)

```

これはセレクションの場合も機能します:

```

1      // $1 contains the primary key of the manager
2      // $2 is a pointer to the resulting array
3
4  QUERY([Employee];[Employee]ID;=$1) // finds the employee whose primary key w
5  APPEND TO ARRAY($2->:[Employee]firstname+" "+[Employee]lastname) // and adds
6
7  QUERY([Employee];[Employee]managerID;=$1:[Employee]ID) // now finds all direct
8
9  If (Records in selection([Employee])>0) // if there are some
10
11    C_LONGINT($i)
12    ARRAY LONGINT($IDS;0)
13    SELECTION TO ARRAY([Employee]ID;$IDS)

```

カスタムウォッチャエリアへの式の追加

ソースコードエリアで選択した式は [カスタムウォッチャエリア](#) にコピーすることができます。

- ソースコードエリア内で評価する式を選択します。
- 次のいずれかの方法をおこないます:
 - 選択したテキストをカスタムウォッチャリヤの式欄へドラッグ & ドロップする。
 - Ctrl+D (Windows) または Cmd+D (macOS) を押す。
 - 選択したテキストを右クリックして、コンテキストメニューから 式ペインにコピー コマンドを選択する。

プログラムカウンター

ソースコードエリアの左マージンにある黄色の矢印は、プログラムカウンターと呼ばれます。これは、実行される次の行を表しています。

デフォルトでは、プログラムカウンター行（実行行とも呼ばれます）がデバッガー内でハイライトされています。[環境設定のメソッドページ](#)において、ハイライトカラーをカスタマイズすることができます。

プログラムカウンターの移動

デバッグのために、呼び出し連鎖のトップにあるメソッド（実行中のメソッド）のプログラムカウンターの位置を変更することができます。これには、黄色の矢印をクリックして目的の行まで上下にドラッグします。

これは、その位置からのトレースや実行を追跡するようにデバッガーに指示しているに過ぎません。カウンターの移動そのものはコードを実行したり、実行をキャンセルしたりしません。すべての現在の設定内容、フィールド、変数などに影響はありません。

たとえば:

```
// ...
If(This condition)
    DO_SOMETHING
Else
    DO_SOMETHING_ELSE
End if
// ...
```

行 `If (This condition)` にプログラムカウンターが設定されているとします。次行に進むボタンをクリックすると、プログラムカウンターが行 `DO SOMETHING ELSE` に直接移動します。しかし、今回トレースしたかったのは `DO_SOMETHING` 行のコードでした。このような場合、プログラムカウンターをその行に移動して実行することができます。

コンテキストメニュー

ソースコードエリアのコンテキストメニューを使って、トレースモードでメソッドを実行する際に便利な機能にアクセスできます:

Debug: P_5

Get Reporting Emps

Expression	Value
> Line Objects	
> Variables	
> Current Form Values	
> Constants	
> Semaphores	
> Processes	
> Tables & Fields	
> Sets	
> Named Selections	
> Information	
> Web	

Call Chain
> Get Reporting Emps

Expression	Value
\$result	Undefined
Form	null
\$2	->\$reportingEmps (Get All Reporting Emps)
\$1	1
Local	

```

1 // $1 contains the primary key of the manager
2 // $2 is a pointer to the resulting array
3
4 QUERY([Employee]ID;=[Employee]ID) // finds the employee whose primary key
5 APPEND TO A
6
7 QUERY([Employee]ID;=[Employee]ID) // now finds all direct reports
8
9 If (Records>0)
10 C_LONGINT($IDS)
11 ARRAY LONGINT $IDS
12 SELECTION
13
14 For ($i;1;Size of array($IDS))
15   Get Reporting Emps ($IDS{$i};$2) // now get all reporting employees of
16   End for
17
18

```

Breakpoint Reached

Context menu for the highlighted code block:

- Goto Definition...
- Search References...
- Copy
- Copy to Expression Pane
- Run to Cursor
- Set Next Statement
- Toggle Breakpoint
- Edit Breakpoint...

- 定義に移動...: 選択された要素の定義に移動します。このコマンドは以下の要素に使用できます:
 - プロジェクトメソッド: 新しいメソッドエディターウィンドウにメソッドの内容を表示します。
 - フィールド: ストラクチャーウィンドウのインスペクターにフィールドプロパティを表示します。
 - テーブル: ストラクチャーウィンドウのインスペクターにテーブルプロパティを表示します。
 - フォーム: フォームエディターにフォームを表示します。
 - 変数 (ローカル、プロセス、インタープロセス、\$n 引数): カレントメソッド内の宣言行を表示、または変数が宣言されたコンパイラーメソッドを表示します。
- 参照検索... (メソッドエディターでも利用可能): 現在の要素が参照されているすべてのメソッドとフォームを検索します。現在の要素とは、選択されているものまたはカーソルが置かれているものをいいます。これにはフィールド、変数、コマンド、文字列等が含まれます。検索結果は、標準の検索結果ウィンドウに表示されます。
- コピー: 選択された式が標準のペーストボードへとコピーされます。
- 式ペインにコピー: 選択された式をカスタムウォッチャーアリアにコピーします。
- カーソルまで実行: プログラムカウンターと選択行の間のコードを実行します。
- 次のステートメントを設定: 現在の行および途中の行を実行せずに、プログラムカウンターを選択行まで移動します。選択行は、ユーザーが実行ボタンのいずれかをクリックした際に実行されます。
- ブレークポイントをトグル (メソッドエディターでも利用可能): 選択行のブレークポイントの有無を切り替えます。これによりメソッドエディターのブレークポイントの有無も切り替わります。
- ブレークポイントを編集... (メソッドエディターでも利用可能): ブレークポイントプロパティダイアログボックスを表示します。ここでおこなわれた変更はメソッドエディターにも反映されます。

次/前を検索

専用のショートカットを使用することで選択された文字列を検索することができます:

- 文字列と一致する次の箇所を検索するには、Ctrl+E (Windows) または Cmd+E (macOS) を使用します。
- 文字列と一致する前の箇所を検索するには、Ctrl+Shift+E (Windows) または Cmd+Shift+E (macOS) を使用します。

この検索は、ソースコードエリアにて少なくとも 1文字以上を選択している場合に実行されます。

ショートカット

この節ではデバッグウィンドウで 利用可能なショートカットをリストしています。

実行制御ツールバーにも [ショートカット](#) が設定されています。

ウォッчエリア & カスタムウォッчエリア

- ウォッчエリア内の項目をダブルクリックすると、その項目がカスタムウォッчエリアにコピーされます。
- カスタムウォッчエリア内でダブルクリックすると、新しい式を作成できます。

ソースコードエリア

- 左マージンをクリックすると、ブレークポイントが設定・削除されます。
- Alt+Shift+クリック (Windows) または Option+Shift+クリック (macOS) により、一時的ブレークポイントが設定されます。
- Alt+クリック (Windows) または Option+クリック (macOS) により、ブレーク編集ウィンドウが表示されます。
- 選択された式や要素をドラック & ドロップして、カスタムウォッчエリアにコピーできます。
- Ctrl+D (Windows) または Command+D (macOS) キーを押すことで、カスタムウォッчエリアに選択テキストがコピーされます。
- Ctrl+E (Windows) または Cmd+E (macOS) を押すと、選択文字列に一致する次の箇所を検索します。
- Ctrl+Shift+E (Windows) または Cmd+Shift+E (macOS) を押すと、選択文字列に一致する前の箇所を検索します。

すべてのエリア

- Ctrl + +/- (Windows) または Command + +/- (macOS) を押すと、可読性を向上させるためにフォントサイズが拡大/縮小します。変更されたフォントサイズはメソッドエディターにも適用され、環境設定に保存されます。
- Ctrl + * (Windows) または Command + * (macOS) を押すと、ウォッчエリアが強制的に更新されます。
- 全エリアでいずれの項目も選択されていない場合に Enterキーを押すと、1行ずつ進みます。
- 項目の値が選択されている場合には、矢印キーでリスト内を移動します。
- 項目を編集中の場合には、矢印キーでカーソルが移動します。Ctrl-A/X/C/V (Windows) または Command-A/X/C/V (macOS) を、編集メニューのすべてを選択/切り取り/コピー/貼り付けコマンドへのショートカットとして使用できます。

ブレークポイントとキャッチコマンド

概要

ブレークポイントとキャッチコマンドは、非常に効率的なデバッグ手法です。どちらも、コードの実行を任意のステップで一時停止させる（まだ表示されていない場合はデバッガーウィンドウを表示させる）という同じ効果があります。

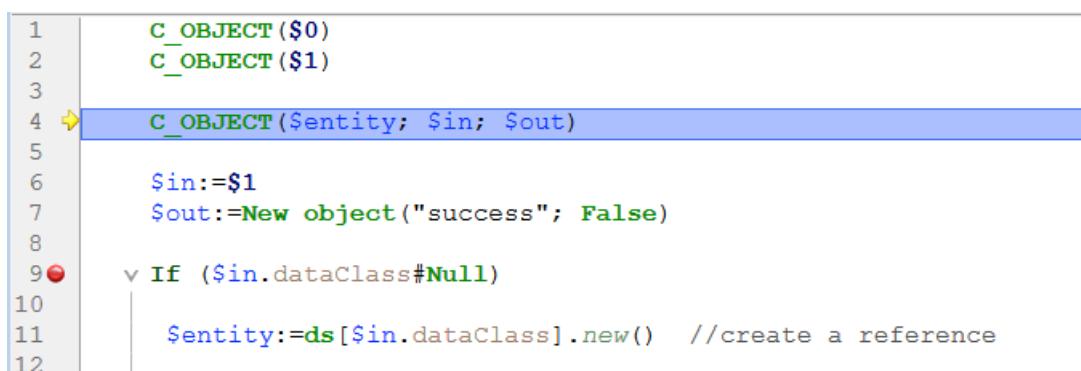
ブレークポイントは、実行を一時停止させたいコードの任意の行に設定します。ブレークポイントには条件を関連付けることができます。

キャッチコマンドは、特定のコマンドが呼び出された時点で、呼び出し元プロセスの実行をトレース開始することができます。

ブレークポイント

ブレークポイントを設定するには、デバッガーまたはメソッドエディターのソースコードエリアの左マージン内をクリックします。

次の図では、ブレークポイント（赤い点）がデバッガー内で、`If ($in.dataClass#Null)` の行に設定されています：



The screenshot shows a Java code editor with the following code:

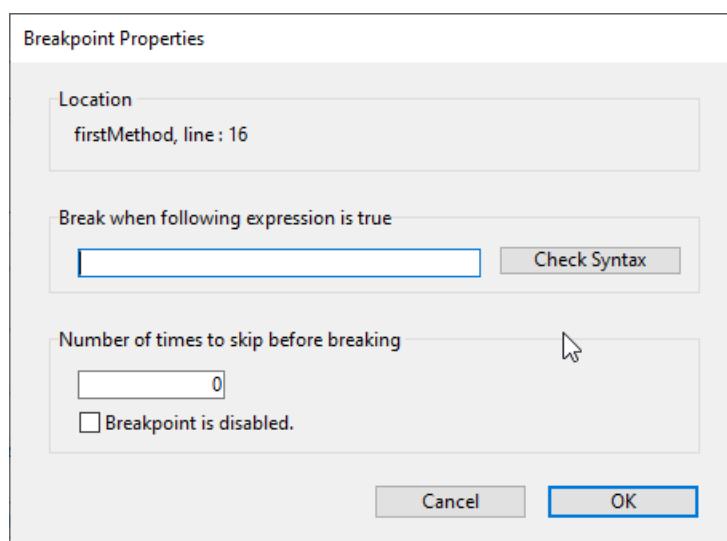
```
1 C_OBJECT($0)
2 C_OBJECT($1)
3
4 C_OBJECT($entity; $in; $out) // Line 4 has a yellow arrow pointing to it
5
6 $in:=$1
7 $out:=New object("success"; False)
8
9 If ($in.dataClass#Null) // Line 9 has a red dot indicating a breakpoint
10
11 $entity:=ds[$in.dataClass].new() //create a reference
12
```

上の状態で **トレース終了** ボタンをクリックすると、ブレークポイントが設定された行まで実行が再開されます。その後、ブレークポイントで示された行は実行されずに、トレースモードへ戻ります。プログラムカウンタより下方の（後に実行される）行にブレークポイントを設定し、トレース終了 ボタンをクリックすると、ブレークポイントまでのメソッドをスキップすることができます。

赤色の点をクリックすると、ブレークポイントは削除されます。

ブレークポイントプロパティ

ブレークポイントプロパティウィンドウを使って、ブレークポイントのふるまいを変更することができます：



このウィンドウはメソッドエディターおよびデバッガーの [ソースコードエリア](#) からアクセスします。次の操作がおこなえます：

- 任意の行を右クリックして、コンテキストメニューから **ブレークポイントを編集...** を選択する。

- 左マージン内で Alt+クリック (Windows) または Option+クリック (macOS) を実行する。

ブレークポイントが既に存在する場合、そのブレークポイントについてのウィンドウが表示されます。それ以外の場合は、ブレークポイントが新規作成され、そのブレークポイントに関するウィンドウを表示します。

プロパティは、次の通りです:

- 場所: メソッド名とブレークポイントが設定されている行番号を示します。
- 次の式が真のときブレーク: True または False を返す 4Dフォーミュラを入力することによって、条件付きブレークポイントを作成することができます。たとえば、Records in selection(\[aTable])=0 と入力すると、テーブル [aTable] のレコードが選択されていない場合に限ってブレークが発生します。ブレークポイントの条件は、[ブレークリスト](#)の 条件 カラムでも確認できます。
- ブレークの前にスキップする回数: ループ構造 (While, Repeat, For) 内、またはループから呼び出されているサブルーチンや関数内のコード行にブレークポイントを設定することができます。
- ブレークポイントが無効です: ブレークポイントが現在は必要でないものの、後で必要になるかもしれない場合には、一時的に無効にしておくことができます。無効なブレークポイントは、点 (·) ではなくダッシュ記号 (-) で表示されます。

リモートデバッグでのブレークポイント

ブレークポイントの一覧はローカルに保存されています。リモートデバッグモードでは、起動したデバッガーがリモート4D だった場合、デバッグセッションの間にはリモートのブレークポイント一覧がサーバーのブレークポイント一覧を一時的に置き換えます。

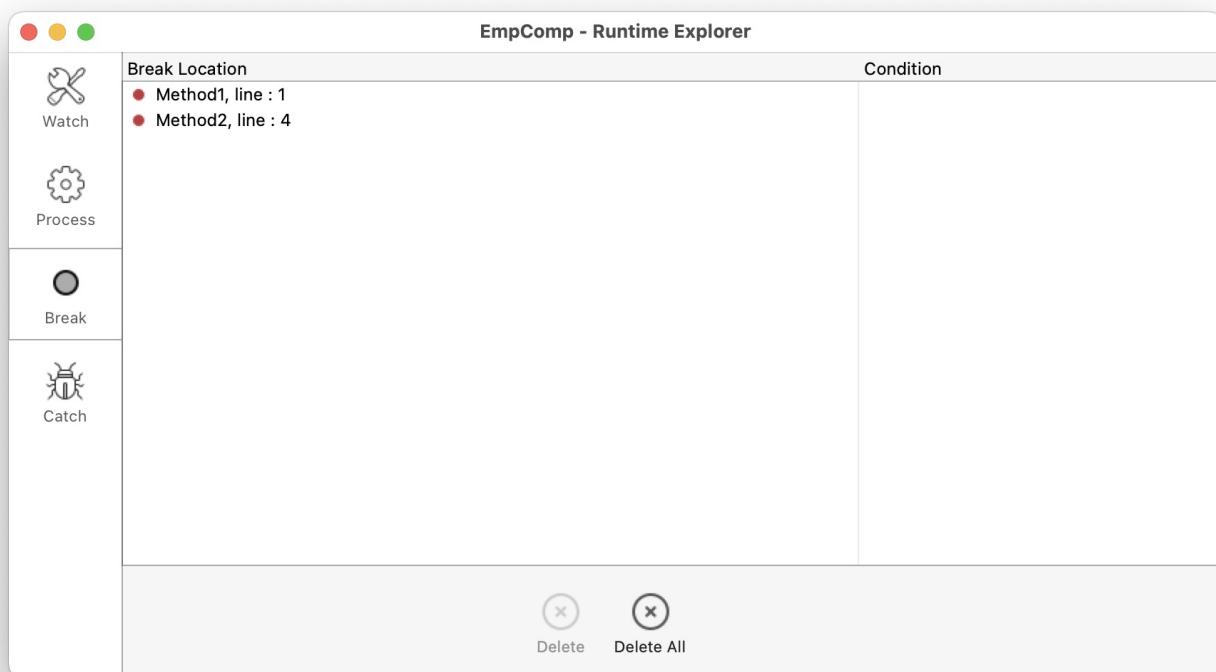
サーバーのブレークポイント一覧は、サーバー上でデバッガーが起動した場合には自動的に復元され使用されます。

ブレークリスト

ブレークリストは、デバッガーウィンドウ又はメソッドエディターで作成したブレークポイントを管理することが出来るランタイムエクスプローラーのページです。ランタイムエクスプローラーの詳細については、[デザインリファレンスマニュアル](#)を参照ください。

ブレークリストのページを開くには:

- 実行 メニューから ランタイムエクスプローラー... を選択します。
- ブレーク タブをクリックして、ブレークリストを表示させます:



このウィンドウを使用して、以下のことが可能です:

- ブレークポイントの 条件 を設定する。
- マージンの赤い点をクリックして、ブレークポイントをそれぞれ有効・無効化する。無効化されたブレークポイントは透明な（薄い赤の）点で表されま

す。

- Delete または Backspace キーを押すか、リスト下の削除 ボタンをクリックして、ブレークポイントを削除する。
- ブレークポイントをダブルクリックして、対象メソッドをエディターで開く。

このウィンドウから新しいブレークポイントを追加することはできません。ブレークポイントは、デバッガーウィンドウかメソッドエディターでのみ設定できます。

コマンドのキャッチ

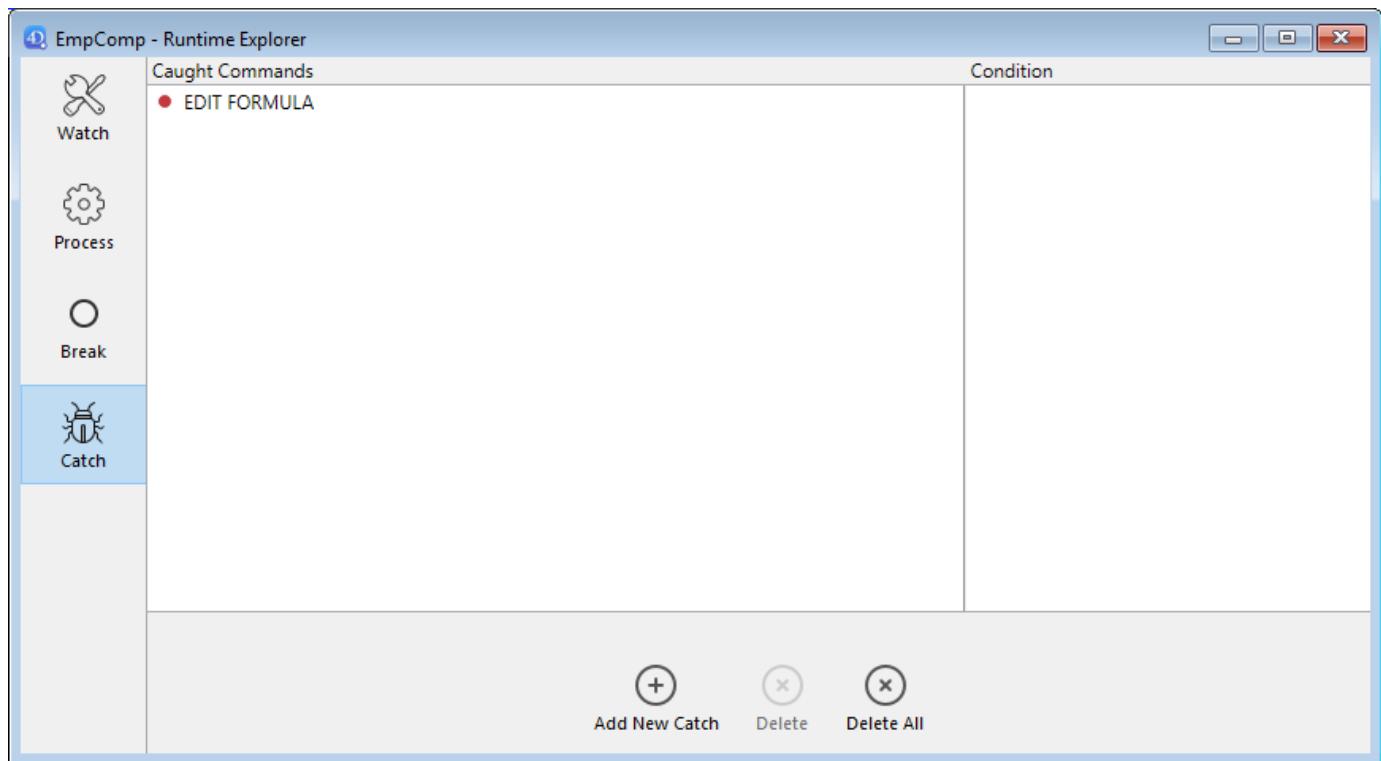
キャッチ コマンドリストは、4Dコマンドの呼び出しを捕捉し、デバッガーウィンドウを表示するよう指示することができるランタイムエクスプローラのページです。特定メソッドの特定行に効果が限定されるブレークポイントとは異なり、キャッチコマンドは、すべてのプロセスおよびメソッドが対象となります。

キャッチコマンド（コマンド捕捉）は、各所にブレークポイントを設定することなく、大きな範囲でトレースをおこなえる便利な方法です。たとえば、いくつかのプロセスを実行した後に、削除すべきでないレコードが削除されてしまう場合には、DELETE RECORD や DELETE SELECTION といったコマンドの処理をキャッチすることにより、調査の範囲を狭めることができます。キャッチ対象のコマンドが呼び出されたたびにデバッガーが起動されるので、問題のレコードが削除されてしまう経緯を調べ、コードの誤った箇所を突き止めることができます。

ブレークポイントとキャッチコマンドは組み合わせて使うことができます。

キャッチコマンドページを開くには：

- 実行 メニューから ランタイムエクスプローラ... を選択します。
- キャッチ タブをクリックすると、キャッチコマンドリストが表示されます：



このページは、実行中にキャッチされるコマンドをリスト表示します。リストは 2つの列で構成されています：

- 左の列には、キャッチするコマンドの有効/無効状況と、コマンド名が表示されます。
- 右の列には、コマンドに関連する条件（あれば）が表示されます。

キャッチするコマンドを新しく追加するには：

- リスト下部にある 新規キャッチを追加 ボタン (+) をクリックします。 ALERT コマンドをデフォルトとして新しいエントリーが追加されます。
- 次に ALERT ラベルをクリックし、キャッチしたいコマンドの名前を入力します。入力したら、Enterキーを押して選択を確定させます。

キャッチコマンドを無効、あるいは有効にするには、コマンドラベルの前にある点 (•) をクリックします。透明な（薄い赤の）点は、キャッチが無効化されていることを表します。

コマンドキャッチの無効化は、削除するのとほぼ同等の効果があります。実行中、デバッガーはほぼ全くと言っていいほどエントリーに時間を使いません。エントリーを無効化することの利点は、それが再び必要になったとき一から作り直さなくて良いという点です。

キヤッチコマンドを削除するには:

1. リスト中のコマンドを選択します。
2. Backspace または Delete キーを押すか、リスト下部にある 削除 ボタンをクリックします。キヤッチコマンドをすべて削除するには、すべてを削除 ボタンをクリックします。

キヤッチコマンドに条件を設定する

1. エントリーの右の列をクリックします。
2. ブール値を返す 4D フォーミュラ (式、コマンドやプロジェクトメソッド) を入力する。

条件を削除するにはフォーミュラを削除します。

条件の設定により、コマンド呼び出し時に特定の条件が満たされている場合にのみ、実行を中止する事ができます。たとえば、`DELETE SELECTION` コマンドのキヤッチに `Records in selection(\[Emp]>10)` という条件を設定した場合、[Emp] テーブルのカレントセレクションが 9 レコード以下の場合には `DELETE SELECTION` コマンドの呼び出しで実行が中断されません。

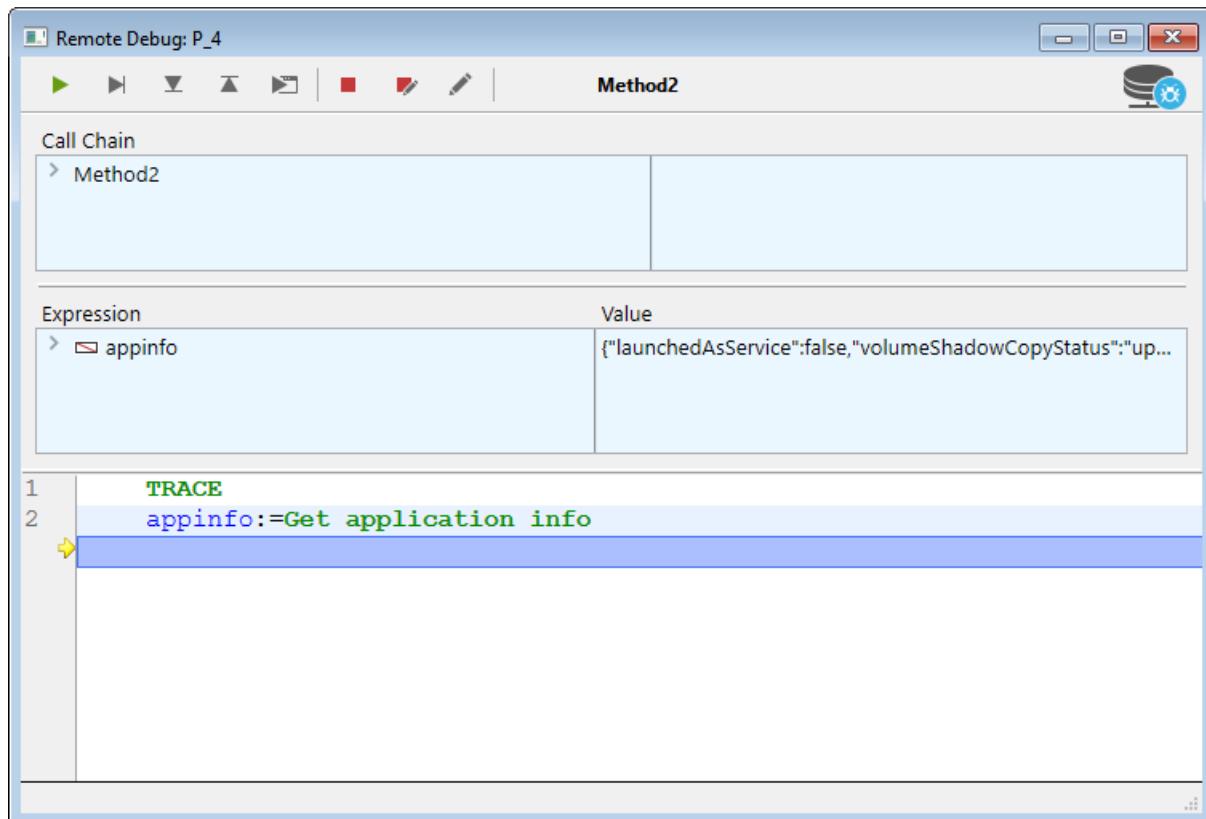
例外のたびに条件を評価することになるため、キヤッチコマンドに条件を追加すると実行速度は遅くなります。その一方で、条件を追加するとデバッグプロセスは早くなります。条件に合致しないオカレンスを、4D が自動的にスキップしていくからです。

リモートマシンからのデバッグ

概要

4Dデータベースが 4D Server 上でインターペリターモードで動いている場合、プロジェクトにログインしているリモート4Dクライアントからサーバーで実行中の 4Dコードをデバッグすることができます。特定のリモートマシンでデバッガーを起動すると、そのリモートマシン上で直接、コードの実行をモニターすることができます。

リモートマシン上で起動された [デバッガー](#) には、サーバーコードのデバッグ中であることを示すサーバーアイコンと青い背景色のデバッグアイコンが表示されるほか、呼び出し連鎖と式のペインの背景が薄っすらと青く色づきます：



この機能は、4D Server がヘッドレスモード ([コマンドラインインターフェース](#) 参照) で実行中、あるいはサーバーマシンへのアクセスが難しい場合などに特に有用です。

有効化済デバッガー

4D Serverアプリケーションのデバッグは、一度に一つのデバッガーのみがおこなえます。これを **有効化済デバッガー** と呼びます。有効化済デバッガーには、以下の 2種類あります：

- ローカルの 4D Serverデバッガー (デフォルト) - サーバーがヘッドレス実行されていない場合
- リモート4Dクライアントのデバッガー - リモートセッションがデザインモードにアクセス可能な場合

有効化済デバッガーは、4D Server が次のいずれかに遭遇した場合に呼び出されます：

- ブレークポイント
- TRACE コマンド
- キャッチコマンド
- エラー

エラーメッセージは、デバッガーが有効化されているマシンに送られるという点に注意してください。これはつまり、リモートデバッガーの場合には、サーバーのエラーメッセージがリモート4Dクライアント上で表示されるということです。

注:

- On Server Startup データベースメソッドで実行されたコードはリモートでデバッグすることができません。これはサーバー側でしかデバッグすることできません。
- デバッガーが有効化されていない場合、実行中のコードがデバッグコマンドによって中断されることはありません。

デバッガーの有効化

インターフォーマンスモードのアプリケーションを起動したとき、デフォルトでは:

- 4D Server がヘッドレス実行中でない場合、デバッガーはサーバー側で有効化されています。
- 4D Server がヘッドレス実行中の場合には、デバッガーは有効化されていない状態です。

4D Server アプリケーションに接続できるリモート 4D クライアントであれば、サーバーのデバッガーを有効化することができます。

リモート 4D クライアントのユーザーセッションは、データベースのデザイン環境へのアクセス権を持っている必要があります。

サーバーのデバッガーをリモート 4D クライアントで有効化するには:

- 4D Server のメニューから、編集 > デバッガを無効化する を選択し、リモートマシンからデバッガーを利用できるようにします (4D Server がヘッドレス実行されている場合、この操作はなにもしません)。
- サーバーに接続されたリモート 4D クライアントから、実行 > リモートデバッガを有効化する を選択します。

有効化に成功した場合 ([有効化リクエストの拒否](#) 参照)、メニュー命令は リモートデバッガを無効化する へと変わります。

これで、サーバーのデバッガーはリモート 4D クライアントで有効化され、以下のタイミングまで有効化されたままです:

- ユーザー session が終了するまで
- ユーザーが リモートデバッガを無効化する を選択するまで

デバッガーを再度サーバー側で有効化するには:

- デバッガーが有効化されているリモート 4D クライアントにおいて、実行 > リモートデバッガを無効化する を選択します。
- 4D Server のメニューから、編集 > デバッガを有効化する を選択します。

サーバー上でデバッガーが有効化されていると (デフォルト)、デバッギングを可能にするため、サーバープロセスはすべて自動的にコオペラティブモードで実行されます。これは、パフォーマンスに大きな影響を与えるかもしれません。サーバーマシン上でデバッギングする必要がない場合は、デバッガーを無効化し、必要に応じてリモートマシンで有効化することが推奨されます。

デバッガを開始時に有効化する

デバッガーは、リモート 4D クライアントまたはサーバーの開始時に自動的に有効化することができます:

- サーバー側の場合 (ヘッドレスモードでなければ)、このオプションは デバッガを開始時に有効化する という名前です。サーバーが開始されると、自動的にデバッガーが有効化されます (デフォルト):

警告: のちにヘッドレスモードで起動されるサーバーにおいてこのオプションが選択されたままの場合、このサーバーのデバッガーは利用できません。

- リモート 4D クライアントでは、このオプションは リモートデバッガを開始時に有効化する という名前です。このオプションが選択されている場合、リモート 4D クライアントは、その後同じ 4D Server データベースに接続するたびに、自動的にリモートデバッガーを有効化しようとします。成功した場合 ([有効化リクエストの拒否](#) 参照)、リモートデバッガーは自動的にリモート 4D クライアントで有効化され、メニュー命令は リモートデバッガを無効化する へと変わります。

この設定はプロジェクトごとに、[.4DPreferences](#) ファイル内にローカル保存されます。

有効化リクエストの拒否

ほかのリモート 4D クライアントまたは 4D Server にてすでに有効化されていた場合、他のマシンでサーバーのデバッガーを有効化することはできません。

別マシンにて有効化済のデバッガーを有効化しようとした場合、その有効化リクエストは拒否され、以下のようなダイアログが表示されます:



このような場合に、デバッガーを有効化するには、以下のどちらかの条件が必要です:

- 有効化済デバッガーを、リモートデバッガを無効化する メニューコマンドでリモート4Dクライアントから外す、あるいは デバッガを無効化する コマンドを使用してサーバーから外す。
- 有効化済デバッガーを使用しているリモート4Dクライアントセッションが閉じられる。

ログファイルの詳細

4Dアプリケーションは、デバッグや実行の最適化のために有用な複数のログファイルを生成することができます。ログは通常 [SET DATABASE PARAMETER](#) あるいは [WEB SET OPTION](#) コマンドのセレクターを使用して開始・停止され、プロジェクトの [Logsフォルダー](#) 内に保存されます。

記録された情報は、問題の検知と修正のためには分析する必要があります。この章では、以下のログファイルの詳細を説明します：

- [4DRequestsLog.txt](#)
- [4DRequestsLog_ProcessInfo.txt](#)
- [HTTPDebugLog.txt](#)
- [4DDebugLog.txt \(標準 & タブ分け\)](#)
- [4DDiagnosticLog.txt](#)
- [4DIMAPLog.txt](#)
- [4DPOP3Log.txt](#)
- [4DSMTPLog.txt](#)
- [ORDA クライアントリクエストのログファイル](#)

サーバーとクライアントの両方においてログファイルが生成可能な場合、サーバー側のログファイル名には "Server" が追加されます。たとえば、"4DRequestsLogServer.txt" のようにです。

これらのログファイルは、デバッグ中に時系列を確立しエントリー間のつながりを分かりやすくするために、いくつかのフィールドを共有しています：

- `sequence_number` : この番号はすべてのデバッグログ間において固有で重複せず、ログファイルに関わらず新しいエントリーごとに増分されるので、オペレーションの厳密な順番を知ることができます。
- `connection_uuid` : 4Dクライアントで作成されサーバーに接続する 4Dプロセスについては、この接続UUID がサーバー側とクライアント側で記録されます。これにより各プロセスを開始したリモートクライアントを簡単に識別することができます。

4DRequestsLog.txt

このログファイルは、コマンドを実行した 4D Serverマシンあるいは 4Dリモートマシンによっておこなわれる標準のリクエストを記録します (Webリクエストを除く)。

このログの開始方法:

- サーバー上:

```
SET DATABASE PARAMETER(4D Server log recording;1)
// サーバーサイド
```

- クライアント上:

```
SET DATABASE PARAMETER(Client Log Recording;1)
// リモートサイド
```

この宣言は [4DRequestsLog_ProcessInfo.txt](#) ログファイルも開始させます。

ヘッダー

このファイルは以下のヘッダーから始まります:

- Logセッション識別子
- アプリケーションをホストしているサーバーのホスト名
- ユーザーログイン名: サーバー上で 4Dアプリケーションを実行したユーザーの OS上のログイン名

内容

各リクエストに対して、以下のフィールドが記録されます:

フィールド名	説明
sequence_number	ログセッション内で固有かつシーケンシャルなオペレーション番号
time	'YYYY-MM-DDTHH:MM:SS.mmm' の ISO 8601フォーマットを使用した日付と時間
systemid	システムID
component	コンポーネント署名 (例: '4SQLS' または 'dbmg')
process_info_index	4DRequestsLog_ProcessInfo.txt ログの "index" フィールドに対応し、リクエストとプロセスのリンクを可能にします。
request	C/SでのリクエストID、あるいは SQLリクエストまたは LOG EVENT メッセージ用のメッセージ文字列
bytes_in	受信したバイト数
bytes_out	送信したバイト数
server_duration exec_duration	ログが生成された場所によって変わります: <ul style="list-style-type: none"> exec_duration (サーバー上で生成された場合) -- サーバーがリクエストを処理するまでにかかった時間 (マイクロ秒単位)。以下の画像の Bから Eまでに相当します。 クライアントがリクエストを送信してサーバーがそれを受け取るまでにかかる時間の概算 (マイクロ秒単位)。以下の画像の Aから Dまでと Eから Hまでに相当します。
write_duration	次のものを送信するのにかかった時間 (μs): <ul style="list-style-type: none"> リクエスト (クライアント上で実行された場合)。以下の画像の Aから Bまでに相当します。 レスポンス (サーバー上で実行された場合)。以下の画像の Eから Fまでに相当します。
task_kind	プリエンプティブかコオペラティブか (それぞれ 'p' と 'c' で表される)
rtt	クライアントがリクエストを送信してサーバーがそれを受け取るまでにかかる時間の概算 (マイクロ秒単位)。以下の画像の Aから Dまでと Eから Hまでに相当します。 <ul style="list-style-type: none"> ServerNet ネットワークレイヤーを使用している場合にのみ計測されます。旧式ネットワークレイヤを使用していた場合には 0 が返されます。 Windows 10、あるいは Windows Server 2016 以前のバージョンの Windowsにおいては、これを呼び出すと 0 が返されます。

リクエストフロー:



このログファイルは、コマンドを実行した 4D Serverマシンあるいは 4Dリモートマシンによって作成された各プロセスについての情報を記録します (Webリクエストを除く)。

このログの開始方法:

- サーバー上:

```
SET DATABASE PARAMETER(4D Server log recording;1) // サーバーサイド
```

- クライアント上:

```
SET DATABASE PARAMETER(Client Log Recording;1) // リモートサイド
```

この宣言は [4DRequestsLog.txt](#) ログファイルも開始させます。

ヘッダー

このファイルは以下のヘッダーから始まります:

- Logセッション識別子
- アプリケーションをホストしているサーバーのホスト名
- ユーザーログイン名: サーバー上で 4Dアプリケーションを実行したユーザーの OS上のログイン名

内容

各プロセスに対して、以下のフィールドが記録されます:

フィールド名	説明
sequence_number	ログセッション内で固有かつシーケンシャルなオペレーション番号
time	"YYYY-MM-DDTHH:MM:SS.mmm" の ISO 8601フォーマットを使用した日付と時間
process_info_index	固有かつシーケンシャルなプロセス番号
CDB4DBaseContext	DB4DコンポーネントデータベースコンテキストUUID
systemid	システムID
server_process_id	サーバー上のプロセスID
remote_process_id	クライアント上のプロセスID
process_name	プロセス名
cID	4D接続の識別子
uID	4Dクライアントの識別子
IP Client	IPv4/IPv6アドレス
host_name	クライアントのホスト名
user_name	クライアント上のユーザーログイン名
connection_uuid	プロセス接続の UUID識別子
server_process_unique_id	サーバー上の固有プロセスID

HTTPDebugLog.txt

このログファイルは、各 HTTPリクエストとそれぞれのレスポンスを rawモードで記録します。ヘッダーを含むリクエスト全体が記録され、オプションでボディ部分も記録することができます。

このログの開始方法:

```
WEB SET OPTION(Web debug log;wdl enable without body) // 他の値も使用可能
```

リクエストとレスポンスの両方に対して以下のフィールドが記録されます:

フィールド名	説明
SocketID	通信に使用されたソケットの ID
PeerIP	ホスト (あるいはクライアント) の IPv4アドレス
PeerPort	ホスト (あるいはクライアント) が使用したポート番号
TimeStamp	(システムが開始されてからの) ミリ秒単位でのタイムスタンプ
ConnectionID	接続UUID (通信に使用された VTCPSocket の UUID)
SequenceNumber	ログセッション内で固有かつシーケンシャルなオペレーション番号

4DDebugLog.txt (標準)

このログファイルは、4Dプログラミングレベルで発生するそれぞれのイベントを記録します。標準モードではイベントの基本的なビューを提供します。

このログの開始方法:

```
SET DATABASE PARAMETER(Debug Log Recording;2)
// 標準、全プロセスを記録

SET DATABASE PARAMETER(Current process debug log recording;2)
// 標準、カレントプロセスのみを記録
```

それぞれのイベントに対して、以下のフィールドが記録されます:

カラム番号	説明
1	ログセッション内で固有かつシーケンシャルなオペレーション番号
2	ISO 8601フォーマットの日付と時間 (YYYY-MM-DDTHH:MM:SS.mmm)
3	プロセスID (p=xx) と固有プロセスID (puid=xx)
4	スタックレベル
5	コマンド名/メソッド名/メッセージ/タスクの開始・停止情報/プラグイン名、イベント、あるいはコールバックUUID または接続UUID
6	ログオペレーションにかかった時間 (ミリ秒単位)

4DDebugLog.txt (タブ分け)

このログファイルは、4Dのプログラミングレベルで発生する各イベントについて、(標準フォーマットに比べて) 追加情報を含めた、タブ分けされたコンパクトなフォーマットで記録します。

このログの開始方法:

```
SET DATABASE PARAMETER(Debug Log Recording;2+4)
// 拡張されたタブ分けされたフォーマット、全プロセスを記録

SET DATABASE PARAMETER(Current process debug log recording;2+4)
// 拡張されたタブ分けされたフォーマット、カレントプロセスのみを記録
```

それぞれのイベントに対して、以下のフィールドが記録されます:

カラム番号	フィールド名	説明
1	sequence_number	ログセッション内で固有かつシーケンシャルなオペレーション番号
2	time	ISO 8601フォーマットの日付と時間 (YYYY-MM-DDTHH:MM:SS.mmm)
3	ProcessID	プロセスID
4	unique_processID	固有プロセスID
5	stack_level	スタックレベル
6	operation_type	<p>ログオペレーションタイプ。この値は絶対値を取ることがあります:</p> <ol style="list-style-type: none"> 1. コマンド 2. メソッド (プロジェクトメソッド、データベースメソッド、等) 3. メッセージ (LOG EVENT コマンドによって送信されたもののみ) 4. プラグインメッセージ 5. プラグインイベント 6. プラグインコマンド 7. プラグインコールバック 8. タスク 9. メンバーメソッド (コレクションまたはオブジェクトに割り当てられているメソッド) <p>スタックレベルを閉じる時には、<code>operation_type</code>、<code>operation</code> および <code>operation_parameters</code> カラムには <code>stack_opening_sequence_number</code> カラムに記録された開始スタックレベルと同じ値が記録されます。たとえば:</p> <ol style="list-style-type: none"> 1. 121 15:16:50:777 5 8 1 2 CallMethod Parameters 0 2. 122 15:16:50:777 5 8 2 1 283 0 3. 123 15:16:50:777 5 8 2 1 283 0 122 3 4. 124 15:16:50:777 5 8 1 2 CallMethod Parameters 0 121 61 <p>1行目と 2行目はスタックレベルを開き、3行目と 4行目はスタックレベルを閉じます。6、7、8カラム目の値は、終了スタックレベル行において繰り返されます。10カラム目にはスタックレベル開始番号、つまり 3行目の 122 と 4行目の 121 が格納されます。</p>
7	operation	以下のいずれかを表す可能性があります (オペレーションタイプによる): <ul style="list-style-type: none"> • ランゲージコマンドID (type=1 の場合) • メソッド名 (type=2 の場合) • pluginIndex;pluginCommand の組み合わせ (type=4、5、6 または 7 の場合)。'3;2' のような形式で格納されます。 • タスク接続UUID (type=8 の場合)
8	operation_parameters	コマンド、メソッド、プラグインに渡された引数
9	form_event	フォームイベント (あれば)。その他の場合には空になります (フォームメソッドまたはオブジェクトメソッド内でコードが実行された場合に使用されると考えて下さい)
10	stack_opening_sequence_number	スタックレベルを閉じる時のみ: 開始スタックレベルに対応するシーケンス番号
11	stack_level_execution_time	スタックレベルを閉じる時のみ: 現在記録されているアクションの経過時間をマイクロ秒単位で表します (上記ログの123 行目と124 行目の 10番目のカラムを参照ください)

4DDiagnosticLog.txt

このログファイルには、アプリケーションの内部オペレーションに関連した複数のイベントが、人間にも読めるように記録されます。[LOG EVENT](#) コマンドを使用することで、カスタムの情報をこのファイルに含めることができます。

このログの開始方法:

```
SET DATABASE PARAMETER(Diagnostic log recording;1) // 記録を開始
```

それぞれのイベントに対して、以下のフィールドが記録されます:

フィールド名	説明
sequenceNumber	ログセッション内で固有かつシーケンシャルなオペレーション番号
timestamp	ISO 8601フォーマットの日付と時間 (YYYY-MM-DDTHH:MM:SS.mmm)
loggerID	任意
componentSignature	任意 - 内部コンポーネント署名
messageLevel	情報、警告、エラーなど
message	ログエントリーの詳細

イベントによって、タスク、ソケットなど様々な他のフィールドを記録に含めることができます。

ファイルを有効化する方法

4DDiagnosticLog.txt ファイルは、ERROR (最も重要) から TRACE (あまり重要でない) まで、異なるレベルのメッセージをログに記録することができます。デフォルトでは、INFO レベルが設定されており、エラーや予期せぬ結果などの重要なイベントのみを記録します (後述参照)。

SET DATABASE PARAMETER コマンドの Diagnostic log level セレクターを使用して、必要に応じてメッセージのレベルを選択することができます。あるレベルを選択すると、その上のレベル (より重要なもの) も暗黙のうちに選択されます。次のレベルが利用可能です:

カラム番号	説明	選択時に次を含みます
ERROR	ログセッション内で固有かつシーケンシャルなオペレーション番号	ERROR
WARN	RFC3339 フォーマットの日付と時間 (yyyy-mm-ddThh:mm:ss.ms)	ERROR, WARN
INFO	4DプロセスID	ERROR, WARN, INFO
DEBUG	固有プロセスID	ERROR, WARN, INFO, DEBUG
TRACE	その他の内部情報 (4Dテクニカルサービス用)	ERROR, WARN, INFO, DEBUG, TRACE

4DSMTPLLog.txt, 4DPOP3Log.txt, および 4DIMAPLog.txt

これらのログファイルは、以下のコマンドを使用して始動された、4Dアプリケーションとメールサーバー (SMTP、POP3、IMAP) 間の通信をそれぞれ記録します:

- SMTP - [SMTP New transporter](#)
- POP3 - [POP3 New transporter](#)
- IMAP - [IMAP New transporter](#)

2種類のログファイルを生成することができます:

- 通常バージョン:

- 4DSMTPLLog.txt, 4DPOP3Log.txt, および 4DIMAPLog.txt と名前がつけられます。
- 添付ファイルは含まれません
- 10MBごとの自動循環ファイルリサイクルを使用します。
- 通常のデバッグ用途を想定しています。

このログを開始するには:

```
SET DATABASE PARAMETER(SMTP Log;1) // SMTPログを開始
SET DATABASE PARAMETER(POP3 Log;1) // POP3ログを開始
SET DATABASE PARAMETER(IMAP Log;1) // IMAPログを開始
```

4D Server: 4D Server 管理ウィンドウ内の [メンテナンスページ](#) の リクエストとデバッグのログを開始 ボタンをクリックします。

このログのパスは `Get 4D file` コマンドによって返されます。

- 拡張バージョン:

- 添付ファイルも含まれます。自動ファイルリサイクルは使用されません。
- カスタムの名前を使用できます。
- 特定の目的のために用意されています。

このログを開始するには:

```
$server:=New object
...
// SMTP
$server.logFile:="MySMTPAuthLog.txt"
$transporter:=SMTP New transporter($server)

// POP3
$server.logFile:="MyPOP3AuthLog.txt"
$transporter:=POP3 New transporter($server)

// IMAP
$server.logFile:="MyIMAPAuthLog.txt"
$transporter:=IMAP New transporter($server)
```

内容

各リクエストに対して、以下のフィールドが記録されます:

カラム番号	説明
1	ログセッション内で固有かつシーケンシャルなオペレーション番号
2	RFC3339 フォーマットの日付と時間 (yyyy-mm-ddThh:mm:ss.ms)
3	4DプロセスID
4	固有プロセスID
5	<ul style="list-style-type: none">SMTP、POP3、または IMAPセッションの切断情報サーバーとクライアント間でやりとりされたデータ。"S <" (SMTP、POP3、または IMAPサーバーから受信したデータ) または "C >" (SMTP、POP3、または IMAPクライアントから送信されたデータ) から始まります: サーバーから送信された認証モードの一覧と選択された認証モード、SMTP、POP3、または IMAPサーバーから報告されたエラー、送信されたメールのヘッダー情報 (通常バージョンのみ) およびメールがサーバー上に保存されているかどうか。SMTP、POP3、または IMAPセッションスタートアップ情報。

ORDAクライアントリクエスト

このログファイルには、リモートマシンが送信する ORDAリクエストが記録されます。ログ情報はメモリか、ディスク上のファイルに書くことができます。このログファイルの名称や保管場所は任意に決めることができます。

このログの開始方法:

```
// リモートマシンで実行します
ds.startRequestLog(File("/PACKAGE/Logs/ordaLog.txt"))
// メモリに送ることもできます
```

ORDAリクエストログにおいてユニークなシーケンス番号を使うには、次のように開始します:

```
// リモートマシンで実行します

SET DATABASE PARAMETER(Client Log Recording;1)
// ログシーケンス番号の有効化

ds.startRequestLog(File("/PACKAGE/Logs/ordaLog.txt"))
// メモリに送ることもできます

SET DATABASE PARAMETER(Client Log Recording;0)
// ログシーケンス番号の無効化
```

各リクエストに対して、以下のフィールドが記録されます:

フィールド名	説明	例題
sequenceNumber	ログセッション内で固有かつシーケンシャルなオペレーション番号	104
url	クライアントの ORDAリクエストURL	"rest/Persons(30001)"
startTime	開始日時 (ISO 8601 フォーマット)	"2019-05-28T08:25:12.346Z"
endTime	終了日時 (ISO 8601 フォーマット)	"2019-05-28T08:25:12.371Z"
duration	クライアント処理時間 (ミリ秒)	25
response	サーバレスポンスオブジェクト	{"status":200,"body": {"__entityModel":"Persons", [...]}}

ログ設定ファイルを使用する

運用環境におけるログ記録を簡単に管理するため、ログ設定ファイル を使用することができます。このファイルは、デベロッパーによってあらかじめ設定されています。一般的には、エンドユーザーに送って、選択してもらおうか、ローカルフォルダーにコピーしてもらいます。一旦有効化されると、ログ設定ファイルは専用仕様のログ記録を開始します。

ファイルを有効化する方法

ログ設定ファイルを有効化する方法はいくつかあります:

- インターフェース付きの 4D Server のメンテナンスページを開き、[ログ設定ファイルを読み込む](#) ボタンをクリックしてファイルを選択します。この場合、設定ファイルには任意の名前を使用することができます。ファイルは、サーバー上で即座に有効化されます。
- ログ設定ファイルを、プロジェクトの [Settingsフォルダー](#) にコピーすることができます。この場合、ファイル名は `logConfig.json` でなくてはなりません。このファイルは、プロジェクトの起動時に有効化されます (クライアント/サーバーのサーバーのみ)。
- ビルドしたアプリケーションでは、`logConfig.json` ファイルを次のフォルダーにコピーできます:
 - Windows: `Users\[userName]\AppData\Roaming\[application]`
 - macOS: `/Users/[userName]/Library/ApplicationSupport/[application]`

スタンドアロン、サーバー、リモート4Dアプリケーションのプロジェクトすべてでログ設定ファイルを有効化するには、次のフォルダーに `logConfig.json` ファイルをコピーします:

- Windows: `Users\[userName]\AppData\Roaming\4D` または `\4D Server`
- macOS: `/Users/[userName]/Library/ApplicationSupport/4D` または `/4D Server`

JSONでの記述

ログ設定ファイルは、次のようなプロパティを持つ `.json` ファイルです:

```
{
    "$schema": "http://json-schema.org/draft-07/schema",
    "title": "Logs Configuration File",
    "description": "A file that controls the state of different types of logs in 4D clients and servers",
    "type": "object",
    "properties": {
        "forceLoggingConfiguration": {
            "description": "Forcing the logs configuration described in the file ingoring changes coming",
            "type": "boolean",
            "default": true
        },
        "requestLogs": {
            "description": "Configuration for request logs",
            "type": "object",
            "properties": {
                "clientState": {
                    "description": "Enable/Disable client request logs (from 0 to N)",
                    "type": "integer",
                    "minimum": 0
                },
                "serverState": {
                    "description": "Enable/Disable server request logs (from 0 to N)",
                    "type": "integer",
                    "minimum": 0
                }
            }
        },
        "debugLogs": {
            "description": "Configuration for debug logs",
            "type": "object",
            "properties": {
                "commandlist": {
                    "description": "Commands to log or not log",
                    "type": "array",
                    "items": {
                        "type": "string"
                    },
                    "minItems": 1,
                    "uniqueItems": true
                },
                "state": {
                    "description": "integer to specify type of debuglog and options",
                    "type": "integer",
                    "minimum": 0
                }
            }
        },
        "diagnosticLogs": {
            "description": "Configuration for debug logs",
            "type": "object",
            "properties": {
                "state": {
                    "description": "Enable/Disable diagnostic logs 0 or 1 (0 = do not record, 1 = record",
                    "type": "integer",
                    "minimum": 0
                }
            }
        },
        "httpDebugLogs": {
            "description": "Configuration for http debug logs"
        }
    }
}
```

```
        "description": "Configuration for http debug logs",
        "type": "object",
        "properties": {
            "level": {
                "description": "Configure http request logs",
                "type": "integer",
                "minimum": 0,
                "maximum": 7
            },
            "state": {
                "description": "Enable/Disable recording of web requests",
                "type": "integer",
                "minimum": 0,
                "maximum": 4
            }
        }
    },
    "POP3Logs": {
        "description": "Configuration for POP3 logs",
        "type": "object",
        "properties": {
            "state": {
                "description": "Enable/Disable POP3 logs (from 0 to N)",
                "type": "integer",
                "minimum": 0
            }
        }
    },
    "SMTPLogs": {
        "description": "Configuration for SMTP logs",
        "type": "object",
        "properties": {
            "state": {
                "description": "Enable/Disable SMTP log recording (form 0 to N)",
                "type": "integer",
                "minimum": 0
            }
        }
    },
    "IMAPLogs": {
        "description": "Configuration for IMAP logs",
        "type": "object",
        "properties": {
            "state": {
                "description": "Enable/Disable IMAP log recording (form 0 to N)",
                "type": "integer"
            }
        }
    },
    "ORDALogs": {
        "description": "Configuration for ORDA logs",
        "type": "object",
        "properties": {
            "state": {
                "description": "Enable/Disable ORDA logs (0 or 1)",
                "type": "integer"
            },
            "filename": {
                "type": "string"
            }
        }
    }
}
```

例題

ログ設定ファイルの例です:

```
{  
    "forceLoggingConfiguration": false,  
    "requestLogs": {  
        "clientState": 1,  
        "serverState": 1  
    },  
    "debugLogs": {  
        "commandList": ["322", "311", "112"],  
        "state": 4  
    },  
    "diagnosticLogs": {  
        "state": 1  
    },  
    "httpDebugLogs": {  
        "level": 5,  
        "state": 1  
    },  
    "POP3Logs": {  
        "state": 1  
    },  
    "SMTPLogs": {  
        "state": 1  
    },  
    "IMAPLogs": {  
        "state": 1  
    },  
    "ORDALogs": {  
        "state": 1,  
        "filename": "ORDALog.txt"  
    }  
}
```

変換タグ

4Dでは、参照を4D変数や式に挿入したり、様々な処理をソーステキスト ("テンプレート") に対して実行したりするための変換タグのセットを用意しています。これらのタグは、ソーステキストが実行されてアウトプットテキストが生成されたときに解釈されます。

4D Webサーバーにおいて [Web テンプレートページ](#) をビルトするにあたって、この原理が使用されます。

これらのタグは原則として HTMLコメント (`<!--#Tag Contents-->`) として挿入します。しかしながら、[xmlに準じた代替シンタックス](#) も一部利用可能です。

複数タイプのタグを混用することも可能です。たとえば、以下のHTML構造は、問題なく実行可能です：

```
<HTML>
<BODY>
<!--#4DSCRIPT/PRE_PROCESS-->    (メソッド呼び出し)
<!--#4DIF (myvar=1)-->    (If 条件)
    <!--#4DINCLUDE banner1.html-->    (サブページ挿入)
<!--#4DENDIF-->    (End if)
<!--#4DIF (mtvar=2)-->
    <!--#4DINCLUDE banner2.html-->
<!--#4DENDIF-->

<!--#4DLOOP [TABLE]-->    (カレントセレクションでのループ)
<!--#4DIF ([TABLE]ValNum>10)-->    (If [TABLE]ValNum>10)
    <!--#4DINCLUDE subpage.html-->    (サブページの挿入)
<!--#4DELSE-->    (Else)
    <B>Value: <!--#4DTEXT [TABLE]ValNum--></B><BR>    (フィールド表示)
<!--#4DENDIF-->
<!--#4DENDLOOP-->    (End for)
</BODY>
</HTML>
```

タグ利用の原則

解析

テンプレートソースの解析は、2つのコンテキストでおこなわれます：

- PROCESS 4D TAGS コマンド使用時：このコマンドは テンプレートに加えて任意の引数を受け入れ、処理の結果であるテキストを返します。
- 4D の統合された HTTPサーバー使用時：WEB SEND FILE (.htm, .html, .shtm, .shtml)、WEB SEND BLOB (text/html型 BLOB)、および WEB SEND TEXT コマンドによって [テンプレートページ](#) を送信、あるいは URL で呼び出します。URL で呼び出す場合、".htm" と ".html" で終わるページは最適化のため解析されません。この場合に HTMLページを解析させるには、終わりを ".shtm" または ".shtml" とする必要があります（例：<http://www.server.com/dir/page.shtm>）。

再起的処理

4Dタグは繰り返し解釈されます。4D は常に変換の結果を解釈しようとし、新しい変換が起きた際にはそれに伴う新しい解釈が実行され、取得結果の変換が必要がなくなるまで繰り返されます。たとえば、以下のようなステートメントがあった場合：

```
<!--#4DHTML [Mail]Letter_type-->
```

もし [Mail]Letter_type テキストフィールド自体にもタグ（たとえば `<!--#4DSCRIPT/m_Gender-->`）が含まれていた場合、このタグは 4DHTMLタグの解釈の後に、それに伴って評価されます。

この強力な原則は、テキスト変換に関連するほとんどの需要を満たすことができます。しかしながら、Webコンテキストにおいて、これは場合によって悪意

のあるコードの侵入を許す可能性があるという点に注意が必要です。これを防ぐ方法については [悪意あるコードの侵入を防止](#) を参照ください。

トークンを使用した識別子

4D のバージョンや言語設定に左右されずに、タグ経由の式の評価が正しくおこなわれることを確実にするため、バージョン間で名前が変わりうる要素（コマンド、テーブル、フィールド、定数）については、トークンシンタックスを使用することが推奨されます。たとえば、`Current time` コマンドを挿入するには、" `Current time:C178` "と入力します。

"." を小数点として使用

`4DTEXT`、`4DHTML`、および `4DEVAL` の 4Dタグで数値表現を評価する際、4D は常にピリオド文字（.）を小数点として使用します。リージョン設定は無視されます。この機能により、4Dの言語設定とバージョンが異なっていてもメンテナンスが容易となり互換性が保たれます。

4DBASE

シンタックス: `<!--#4DBASE folderPath-->`

`<!--#4DBASE -->` タグは `<!--#4DINCLUDE-->` タグで使用されるワーキングディレクトリを指定します。

Webページ内で呼び出されると、`<!--#4DBASE -->` タグは同ページ内あとに続くすべての `<!--#4DINCLUDE-->` 呼び出しのディレクトリを変更します（次の `<!--#4DBASE -->` があるまで）。組み込まれたファイル内で `<!--#4DBASE -->` フォルダーが変更されると、親のファイルから元となる値を取得します。

`folderPath` 引数には現在のページに対する相対パスを指定し、パスは "/" で終わっていなければなりません。また、指定フォルダーは Webフォルダ内になければなりません。

"WEBFOLDER" キーワードを渡すと、(そのページに対して相対の) デフォルトパスに戻されます。

以下のように、各呼び出しごとに相対パスを指定したコードは：

```
<!--#4DINCLUDE subpage.html-->
<!--#4DINCLUDE folder/subpage1.html-->
<!--#4DINCLUDE folder/subpage2.html-->
<!--#4DINCLUDE folder/subpage3.html-->
<!--#4DINCLUDE ../folder/subpage.html-->
```

以下のコードと同一です：

```
<!--#4DINCLUDE subpage.html-->
<!--#4DBASE folder-->
<!--#4DINCLUDE subpage1.html-->
<!--#4DINCLUDE subpage2.html-->
<!--#4DINCLUDE subpage3.html-->
<!--#4DBASE ../folder-->
<!--#4DINCLUDE subpage.html-->
<!--#4DBASE WEBFOLDER-->
```

たとえば、ホームページのディレクトリを設定する場合：

```
/* Index.html */
<!--#4DIF LangFR=True-->
    <!--#4DBASE FR-->
<!--#4ELSE-->
    <!--#4DBASE US-->
<!--#4ENDIF-->
<!--#4DINCLUDE head.html-->
<!--#4DINCLUDE body.html-->
<!--#4DINCLUDE footer.html-->
```

上で組み込まれる "head.html" ファイル内でカレントフォルダーが `<!--#4DBASE -->` を使用して変更されても、"index.html" 内では変更されません：

```
/* Head.htm */
/* ここでのワーキングディレクトリはインクルードされるファイルに対して相対的 (FR/ または US/) */
<!--#4DBASE Styles-->
<!--#4DINCLUDE main.css-->
<!--#4DINCLUDE product.css-->
<!--#4DBASE Scripts-->
<!--#4DINCLUDE main.js-->
<!--#4DINCLUDE product.js-->
```

4DCODE

シンタックス: `<!--#4DCODE codeLines-->`

`4DCODE` タグを使用すると、複数行の4Dコードのブロックをテンプレートに挿入できます。

"`<!--#4DCODE`" シークエンスとそれに続くスペース、CRまたはLF文字が検知されると、4D は次の "`-->`" シークエンスまでのコードを解釈します。コードブロック自体はキャリッジリターンもラインフィードも、あるいはその両方も含むことができ、4D によってシーケンシャルに解釈されます。

たとえば、以下のようにテンプレートに書くことができます：

```
<!--#4DCODE
// パラメーターの初期化
C_OBJECT:C1216($graphParameters)
OB SET:C1220($graphParameters;"graphType";1)
$graphType:=1
//...ここにコードを書きます
If(OB Is defined:C1231($graphParameters;"graphType"))
    $graphType:=OB GET:C1224($graphParameters;"graphType")
    If($graphType=7)
        $nbSeries:=1
        If($nbValues>8)
            DELETE FROM ARRAY:C228 ($yValuesArrPtr{1}=>;9;100000)
            $nbValues:=8
        End if
    End if
End if
-->
```

`4DCODE` タグの機能は以下の通りです：

- `TRACE` コマンドがサポートされています。これは 4Dデバッガーを起動するので、テンプレートコードをデバッグすることができます。
- エラーは標準のエラーダイアログを表示します。これを使って、ユーザーはコードの実行を中止したりデバッグモードに入ったりすることができます。
- `<!--#4DCODE` と `-->` の間のテキストは改行され、どのような改行コードでも受け取ります (cr, lf, または crlf)。
- テキストは `PROCESS 4D TAGS` を呼び出したデータベースのコンテキストにてトークナイズされます。これは、たとえばプロジェクトメソッドの認識等において重要です。 [公開オプション: 4DタグとURL\(4DACTION...\)](#) メソッドプロパティは考慮されません。
- テキストが常に English-US設定であったとしても、4Dのバージョン間においてコマンドや定数名が改名されることによる問題を避けるため、コマンド名や定数名はトーカンシンタックスを使用することが推奨されています。

4DCODE タグがあらゆる 4Dランゲージコマンドおよびプロジェクトメソッドを呼び出せるという事実は、とくにデータベースが HTTP経由で使用可能な場合等に、セキュリティ上の問題になり得ます。しかしながら、タグはサーバー側のコードをテンプレートファイルから実行するため、タグそのものはセキュリティ上の問題になりません。このようなコンテキストにおいては、あらゆる Webサーバーと同様に、セキュリティは主にサーバーファイルへのリモートアクセスレベルにおいて管理されています。

4DEACH と 4DENDEACH

シンタックス: <!--#4DEACH variable in expression--> <!--#4DENDEACH-->

<!--#4DEACH--> コメントは、*expression* に含まれるすべての要素に対して処理を繰り返します。各要素は *variable* に代入され、その型は *expression* の型に依存します。

<!--#4DEACH--> コメントは 3種類の *expression* を対象に反復処理をおこなうことができます:

- コレクション: コレクションの各要素をループします
- エンティティセレクション: エンティティセレクションの各エンティティをループします
- オブジェクト: オブジェクトの各プロパティをループします

ループの数は開始時に評価され、処理中に変化することはありません。ループ中に項目を追加・削除することは、繰り返しの不足・重複を引き起こすことがあるため、一般的には推奨されません。

<!--#4DEACH item in collection-->

このシンタックスは、コレクションの各要素を対象に反復処理をおこないます。 <!--#4DEACH --> と <!--#4DENDEACH--> の間に書かれたコードが、各コレクション要素について繰り返されます。

item はコレクション要素と同じ型の変数です。

コレクションの 要素はすべて同じ型 でなくてはなりません。そうでない場合には、*item* 変数に別の型の値が代入されたときにエラーが生成されます。

ループの回数はコレクションの要素数に基づいています。各繰り返しにおいて、*item* 変数には、コレクションの合致する要素が自動的に代入されます。このとき、以下の点に注意する必要があります:

- item* 変数がオブジェクト型あるいはコレクション型であった場合 (つまり *expression* がオブジェクトのコレクション、あるいはコレクションのコレクションであった場合)、この変数を変更すると自動的にコレクションの対応する要素も変更されます (オブジェクトとコレクションは同じ参照を共有しているからです)。変数がスカラー型である場合、変数のみが変更されます。
- item* 変数には、コレクションの先頭要素の型が設定されます。コレクション要素のどれか一つでも、変数と異なる型のものがあった場合、エラーが生成され、ループは停止します。
- コレクションが Null 値の要素を格納していたとき、*item* 変数の型が Null 値をサポートしない型 (倍長整数変数など) であった場合にはエラーが生成されます。

例題: スカラー値のコレクション

getNames は文字列のコレクションを返すメソッドです。このメソッドは [公開オプション: 4DタグとURL](#) が有効になっています。

```
<table class="table">

<tr><th>Name</th></tr>

<!--#4DEACH $name in getNames-->
<tr>
    <td><!--#4DTEXT $name--></td>
</tr>
<!--#4DENDEACH-->
</table>
```

例題: オブジェクトのコレクション

getSalesPersons はオブジェクトのコレクションを返すメソッドです。

```





```

<!--#4DEACH entity in entitySelection-->

このシンタクスは、エンティティセレクションの各エンティティを対象に反復処理をおこないます。 <!--#4DEACH --> と <!--#4DENDEACH--> の間に書かれたコードが、エンティティセレクションの各エンティティについて繰り返されます。

entity は Entityクラスのオブジェクト変数です。

ループの回数はエンティティセレクション内のエンティティの数に基づきます。各繰り返しにおいて、*entity* オブジェクト変数には、エンティティセレクションの合致するエンティティが自動的に代入されます。

例題: html のテーブル

```





```

例題: PROCESS 4D TAGS

```

var customers : cs.CustomersSelection
var $input; $output : Text

customers:=ds.Customers.all()
$input:="<!--#4DEACH $cust in customers-->"
$input:=$input+"<!--#4DTEXT $cust.name -->"+Char(Carriage return)
$input:=$input+"<!--#4DENDEACH-->"
PROCESS 4D TAGS($input; $output)
TEXT TO DOCUMENT("customers.txt"; $output)

```

<!--#4DEACH property in object-->

このシンタクスは、オブジェクトの各プロパティを対象に反復処理をおこないます。 <!--#4DEACH --> と <!--#4DENDEACH--> の間に書かれたコードが、各オブジェクトプロパティについて繰り返されます。

property は現在処理中のプロパティ名が自動代入されたテキスト変数です。

オブジェクトのプロパティは作成順に処理されていきます。ループ中、プロパティをオブジェクトに追加/削除することが可能ですが、その場合でも残りのループ回数は、オブジェクトの元のプロパティ数に基づいています。

例題: オブジェクトのプロパティ

`getGamers` は、ゲームスコアを管理するために ("Mary"; 10; "Ann"; 20; "John"; 40) のようなオブジェクトを返すプロジェクトメソッドです。

```
<table class="table">
<!--#4D CODE
$gamers:=getGamers
-->

<tr><th>Gamers</th><th>Scores</th></tr>

<!--#4D EACH $key in $gamers-->
<tr>
<td><!--#4D TEXT $key--></td>
<td><!--#4D TEXT $gamers[$key]--></td>
</tr>
<!--#4D EACH-->
</table>
```

4DEVAL

シンタクス: `<!--#4DEVAL expression-->`

代替シンタクス: `$4DEVAL(expression)`

`4DEVAL` タグを使用すると、4Dの変数や式を評価できます。`4DHTML` タグのように、`4DEVAL` タグはテキストを返す際にHTML特殊文字をエスケープしません。しかしながら、`4DHTML` や `4DTEXT` と異なり、`4DEVAL` は有効な 4D宣言であればどれでも実行することができます（値を返さない代入や式も含まれます）。

たとえば、以下の様なコードを実行することができます:

```
$input:="<!--#4DEVAL a:=42-->" // 代入
$input:=$input+"<!--#4DEVAL a+1-->" // 計算
PROCESS 4D TAGS($input;$output)
// $output = "43"
```

解釈エラーの場合、" `<!--#4DEVAL expr-->: ## エラー # エラーコード` " というテキストが挿入されます。

セキュリティ上の理由から、[悪意あるコードの侵入・挿入](#) を防ぐために、アプリケーション外から導入されたデータを処理するときには `4DTEXT` タグの使用が推奨されます。

4DHTML

シンタクス: `<!--#4DHTML expression-->`

代替シンタクス: `$4DHTML(expression)`

`4DTEXT` タグ同様、このタグを使用すると、4Dの変数や値を返す式を HTML式として挿入できます。一方 `4DTEXT` タグとは異なり、このタグは HTML特殊文字(例: ">")をエスケープしません。

たとえば、4Dタグを使用して 4Dのテキスト変数 `myvar` を処理した結果は以下の様になります:

myvar の値	タグ	戻り値
myvar:=""	<!--#4DTEXT myvar-->	
myvar:=""	<!--#4DHTML myvar-->	

解釈エラーの場合、" <!--#4DHTML myvar-->: ## エラー # エラーコード " というテキストが挿入されます。

セキュリティ上の理由から、[悪意あるコードの侵入・挿入](#)を防ぐために、アプリケーション外から導入されたデータを処理するときには [4DTEXT](#) タグの使用が推奨されます。

4DIF, 4ELSE, 4ELSEIF と 4ENDIF

シンタックス: <!--#4DIF expression--> { <!--#4ELSEIF expression2-->...<!--#4ELSEIF expressionN--> } { <!--#4ELSE--> } <!--#4ENDIF-->

<!--#4ELSEIF--> (任意), <!--#4ELSE--> (任意) および <!--#4ENDIF--> コメントと共に使用することで、<!--#4DIF expression--> コメントはコードの一部に条件分岐を実行させることを可能にします。

expression はブール値を返す有効な 4D式です。式は括弧の中に記述され、4Dのシンタクスルールに準拠していかなければなりません。

<!--#4DIF expression--> ... <!--#4ENDIF--> は複数レベルでネストできます。4Dと同じく、それぞれの <!--#4DIF expression--> には対応する <!--#4ENDIF--> がなければなりません。

解釈エラーの場合、<!--#4DIF --> と <!--#4ENDIF--> の間のコンテンツの代わりに、" <!--#4DIF expression--> : ブール式が必要です" というテキストが挿入されます。同様に、<!--#4DIF --> が同じ数の <!--#4ENDIF--> で閉じられていない場合、<!--#4DIF --> と <!--#4ENDIF--> の間のコンテンツの代わりに " <!--#4DIF expression--> : 4DENDIFが必要です" というテキストが挿入されます。

<!--#4ELSEIF--> タグを使用すると、数に制限なく条件をテストできます。最初に `true` と判定されたブロック内にあるコードだけが実行されます。`true` ブロックがなく、<!--#4ELSE--> もない場合には、なにも実行されません。<!--#4ELSE--> タグは、最後の <!--#4ELSEIF--> の後に記述できます。それまでの条件がすべて `false` の場合、<!--#4ELSE--> ブロックの文が実行されます。

以下の2つのコードは同等です。

4ELSE のみを使用する場合:

```
<!--#4DIF Condition1-->
/* Condition1 が true の場合*/
<!--#4ELSE-->
    <!--#4DIF Condition2-->
        /* Condition2 が true の場合*/
    <!--#4ELSE-->
        <!--#4DIF Condition3-->
            /* Condition3 が true の場合 */
        <!--#4ELSE-->
            /*いずれの条件も true でない場合*/
        <!--#4ENDIF-->
    <!--#4ENDIF-->
<!--#4ENDIF-->
```

同じ内容を 4ELSEIF タグを使用して記述した場合:

```
<!--#4DIF Condition1-->
/* Condition1 が true の場合*/
<!--#4DELSEIF Condition2-->
/* Condition2 が true の場合*/
<!--#4DELSEIF Condition3-->
/* Condition3 が true の場合 */
<!--#4ELSE-->
/* いずれの条件も true でない場合*/
<!--#4ENDIF-->
```

スタティックな HTML ページに書かれたこの例題のコードは、`vname#""` 式の結果に応じ、異なるラベルを表示します：

```
<BODY>
...
<!--#4DIF (vname#"")-->
Names starting with <!--#4DTEXT vname-->.
<!--#4ELSE-->
No name has been found.
<!--#4ENDIF-->
...
</BODY>
```

この例題は接続したユーザーによって異なるページを返します：

```
<!--#4DIF LoggedIn=False-->
<!--#4DINCLUDE Login.htm -->
<!--#4DELSEIF User="Admin" -->
<!--#4DINCLUDE AdminPanel.htm -->
<!--#4DELSEIF User="Manager" -->
<!--#4DINCLUDE SalesDashboard.htm -->
<!--#4ELSE-->
<!--#4DINCLUDE ItemList.htm -->
<!--#4ENDIF-->
```

4DINCLUDE

シンタックス: `<!--#4DINCLUDE path-->`

このタグは主に、ある (`path` で指定された) HTML ページを別の HTML ページに含めるためにデザインされました。デフォルトで、HTML ページのボディー部、つまり `<body>` と `</body>` タグの間の内容だけが統合されます (`body` タグは含まれません)。これにより、ヘッダーに含まれるメタタグ関連の衝突が回避されます。

しかし、指定された HTML ページ中に `<body>` `</body>` タグがない場合、ページ全体が統合されます。この場合、メタタグの整合性を管理するのには開発者の役割です。

`<!--#4DINCLUDE -->` コメントは、テスト (`<!--#4DIF-->`) やループ (`<!--#4DLOOP-->`) と使用するととても便利です。条件に基づきあるいはランダムにバナーなどを挿入する便利な方法です。このタグを使用してページをインクルードするとき、拡張子にかかわらず、4D は呼び出されたページを解析してから、内容を `4DINCLUDE` 呼び出し元のページに挿入します。

`<!--#4DINCLUDE -->` コメントで挿入されたページは、URL で呼ばれたページや `WEB SEND FILE` コマンドで送信されたページと同じように、Web サーバーキャッシュにロードされます。

`path` には、挿入するドキュメントのパスを記述します。警告: `4DINCLUDE` を呼び出す場合、パスは解析される親ドキュメントを起点とした相対パスです。フォルダ区切り文字にはスラッシュ (/) を使用し、レベルをさかのぼるには 2 つのドット (..) を使用します (HTML シンタックス)。`PROCESS 4D TAGS` コマンドで `4DINCLUDE` タグを使用する場合のデフォルトフォルダーはプロジェクトフォルダーです。

`4DINCLUDE` タグで使用されるデフォルトフォルダーは `<!--#4DBASE -->` タグを使って変更できます。

ページ内で使用できる `<!--#4DINCLUDE path-->` 数に制限はありません。しかし `<!--#4DINCLUDE path-->` の呼び出しは 1 レベルのみ有効です。つまり、たとえば `mydoc1.html` ページに `<!--#4DINCLUDE mydoc2.html-->` によって挿入される `mydoc2.html` がある場合、そのボディ内でさらに `<!--#4DINCLUDE mydoc3.html-->` を使うことはできません。さらに、4Dはインクルードが再帰的かどうかを確認します。

エラーの場合、" `<!--#4DINCLUDE path-->` : ドキュメントを開けません" というテキストが挿入されます。

例:

```
<!--#4DINCLUDE subpage.html-->
<!--#4DINCLUDE folder/subpage.html-->
<!--#4DINCLUDE ../folder/subpage.html-->
```

4DLOOP と 4DENDLOOP

シンタックス: `<!--#4DLOOP condition--> <!--#4DENDLOOP-->`

このコメントを使用して、条件を満たす間、コードの一部を繰り返すことができます。繰り返し部のコードは `<!--#4DLOOP-->` と `<!--#4DENDLOOP-->` で挟まれます。

`<!--#4DLOOP condition--> ... <!--#4DENDLOOP-->` ブロックはネストできます。4Dと同じく、それぞれの `<!--#4DLOOP condition-->` には対応する `<!--#4DENDLOOP-->` がなければなりません。

5種類の条件を使用できます:

```
<!--#4DLOOP [table]-->
```

このシンタックスは、カレントプロセスのカレントセレクションに基づき、指定したテーブルのレコード毎にループします。カレントセレクションレコード毎に、2つのコメントの間のコードは繰り返されます。

テーブルを条件として `4DLOOP` タグが使用されると、レコードが "読み取り専用" モードでロードされます。

以下のコードは:

```
<!--#4DLOOP [People]-->
<!--#4DTEXT [People]Name--> <!--#4DTEXT [People]Surname--><BR>
<!--#4DENDLOOP-->
```

4Dランゲージで表すと以下のとおりです:

```
FIRST RECORD([People])
While(Not(End selection([People])))
...
NEXT RECORD([People])
End while
```

```
<!--#4DLOOP array-->
```

このシンタックスは、配列項目ごとにループします。2つのコメントの間のコードが繰り返されるたびに、配列のカレント項目がインクリメントされます。

このシンタックスで二次元配列を使用することはできません。この場合には、代わりにメソッドを条件としたループをネストして使用します。

以下のコードは:

```
<!--#4DLOOP arr_names-->
<!--#4DTEXT arr_names{arr_names}--><BR>
<!--#4DENDLOOP-->
```

4Dランゲージで表すと以下のとおりです:

```
For($Elem;1;Size of array(arr_names))
    arr_names:=$Elem
    ...
End for
```

```
<!--#4DLOOP method-->
```

このシンタクスでは、メソッドが `true` を返す間ループがおこなわれます。メソッドは、倍長整数タイプの引数を受け取ります。まずメソッドは引数 0 を渡されます。これは（必要に応じて）初期化ステージとして使用できます。その後、`true` が返されるまで 1, 2, 3 と渡される引数値がインクリメントされます。

セキュリティのため、Webプロセス内では、`On Web Authentication` データベースメソッドが初期化ステージ（引数に0が渡されて実行される）の前に一度呼び出されます。認証に成功すると、初期化に進みます。

コンパイルのため、`C_BOOLEAN($0)` と `C_LONGINT($1)` が必ず宣言されていなければなりません。

以下のコードは:

```
<!--#4DLOOP my_method-->
<!--#4DTEXT var--> <BR>
<!--#4DENDLOOP-->
```

4Dランゲージで表すと以下のとおりです:

```
If(AuthenticationWebOK)
    If(my_method(0))
        $counter:=1
        While(my_method($counter))
            ...
            $counter:=$counter+1
        End while
    End if
End if
```

`my_method` は以下のようになります:

```
C_LONGINT($1)
C_BOOLEAN($0)
If($1=0) `Initialisation
    $0:=True
Else
    If($1<50)
        ...
        var:...
        $0:=True
    Else
        $0:=False `Stops the loop
    End if
End if
```

```
<!--#4DL0OP expression-->
```

このシンタクスでは、`4DL0OP` タグは `expression` に指定した式が `true` を返す間ループがおこなわれます。式は有効なブール式であればよく、無限ループを防ぐために、ループごとに評価される変数部分を含んでいる必要があります。

たとえば、以下のコードは：

```
<!--#4DEVAL $i:=0-->
<!--#4DL0OP ($i<4)-->
<!--#4DEVAL $i-->
<!--#4DEVAL $i:=$i+1-->
<!--#4DENDL0OP-->
```

以下の結果を生成します：

```
0
1
2
3
```

```
<!--#4DL0OP pointerArray-->
```

この場合、`4DL0OP` タグは配列のときと同じように振るまいります：ポインターによって参照された配列の要素ごとにループを繰り返します。カレントの配列要素は、コードが繰り返される度に増加していきます。

このシンタクスは `PROCESS 4D TAGS` コマンドに対して配列ポインターを渡した場合に有用です。

例：

```
ARRAY TEXT($array;2)
$array{1}:="hello"
$array{2}:="world"
$input:="<!--#4DEVAL $1--&gt;"
$input:=$input+"<!--#4DL0OP $2--&gt;" 
$input:=$input+"<!--#4DEVAL $2--&gt;{$2--&gt;}--&gt; "
$input:=$input+"<!--#4DENDL0OP--&gt;" 
PROCESS 4D TAGS($input;$output;"elements = ";-&gt;$array)
// $output = "elements = hello world "</pre>
```

解釈エラーの場合、`<!--#4DL0OP -->` と `<!--#4DENDL0OP-->` の間のコンテンツの代わりに "`<!--#4DL0OP expression-->` : エラーの説明" というテキストが挿入されます。

以下のメッセージが表示されます：

- 予期しない式のタイプ (標準のエラー);
- テーブル名が正しくありません (テーブル名のエラー);
- 配列が必要です (変数が配列でないか、二次元配列が指定された);
- メソッドが存在しません;
- シンタクスエラー (メソッド実行時);
- アクセス権エラー (テーブルやメソッドにアクセスする権限がない);
- 4DENDL0OP が必要です (`<!--#4DL0OP -->` が対応する `<!--#4DENDL0OP-->` で閉じられていない)。

4DSCRIPT/

シンタクス: `<!--#4DSCRIPT/MethodName/MyParam-->`

`4DSCRIPT` タグは、テンプレートを処理する際に 4Dメソッドを実行することを可能にします。`<!--#4DSCRIPT/MyMethod/MyParam-->` タグが

HTMLコメントとしてページに現れると、`MyMethod` メソッドが `$1` に `Param` を受け取って実行されます。

タグが Webプロセスのコンテキストにおいて呼び出された場合、Webページがロードされると、4Dは `On Web Authentication` データベースメソッドを (存在すれば) 呼び出します。このメソッドが `true` を返すと、4Dはメソッドを実行します。

メソッドは `$0` にテキストを返す必要があります。文字列が文字コード 1 (つまり、`Char(1)` のこと) から始まっていると、それは HTMLソースとして扱われます (4DHTML と同じ原則)。

たとえば、次のコメントをテンプレートWebページに挿入したとしましょう: "今日の日付は<!--#4DSCRIPT/MYMETH/MYPARAM-->" 。ページをロードする際、4Dは `On Web Authentication` データベースメソッドを (存在すれば) 呼び出し、そして `MYMETH` メソッドの `$1` に文字列 "/MYPARAM" を引数として渡して呼び出します。メソッドは `$0` にテキストを返します (たとえば "21/12/31")。"今日の日付は <!--#4DSCRIPT/MYMETH/MYPARAM-->" というコメントの結果は "今日の日付は 21/12/31" となります。

`MYMETH` メソッドは以下のとおりです:

```
//MYMETH  
C_TEXT($0;$1) // これらのパラメーターは常に宣言する必要があります  
$0:=String(Current date)
```

`4DSCRIPT` から呼び出されるメソッドは、インターフェース要素 (`DIALOG` , `ALERT` など) を呼び出してはいけません。

4Dはメソッドを見つけた順に実行するため、ドキュメントの後の方で参照される変数の値を設定するメソッドを呼び出すことも可能です。モードは関係ありません。テンプレートには必要なだけ `<!--#4DSCRIPT...-->` コメントを記述できます。

4DTEXT

シンタクス: `<!--#4DTEXT expression-->`

代替シンタクス: `$4DTEXT(expression)`

タグ `<!--#4DTEXT VarName-->` を使用して 4D変数や値を返す式への参照を挿入できます。たとえば、(HTMLページ内にて) 以下のように記述すると:

```
<P><!--#4DTEXT vtSiteName--> へようこそ！</P>
```

4D変数 `vtSiteName` の値が HTMLページに送信時に挿入されます。値はテキストとして挿入されます。">"のようなHTMLの特殊文字は、自動的にエスケープされます。

4DTEXT タグを使用して、4D式も挿入できます。たとえば、フィールドの値を直接挿入できるほか (`<!--#4DTEXT [tableName]fieldName-->`)、配列要素の値も挿入できますし (`<!--#4DTEXT tabarr{1}-->`)、値を返すメソッドも使用できます (`<!--#4DTEXT mymethod-->`)。式の変換には、変数の場合と同じルールが適用されます。さらに、式は 4Dのシンタクスルールに適合していなければなりません。

セキュリティ上の理由から、[悪意あるコードの侵入・挿入](#)を防ぐために、アプリケーション外から導入されたデータを処理するときには、このタグの使用が推奨されます。

解釈エラーの場合、"`<!--#4DTEXT myvar--> : ## エラー # エラーコード`" というテキストが挿入されます。

- プロセス変数を使用する必要があります。
- ピクチャーフィールドの内容を表示できます。しかし、ピクチャー配列の要素を表示することはできません。
- 4D式を使用して、オブジェクトフィールドの中身を表示させることができます。たとえば、次の様に記述します: `<!--#4DTEXT OB Get:C1224([Rect]Desc;"color\")-->` .
- 通常はテキスト変数を使用します。しかし、BLOB変数を使用することもできます。この場合、長さ情報なしのテキストBLOBを使用します。

4dtext, 4dhtml, 4deval の代替シンタクス

いくつかの既存の 4D変換タグは、\$-ベースのシンタックスを使用して表現することができます：

\$4dtag (expression)

という表記を次の代わりに使用することができます：

```
<!--#4dtag expression-->
```

この代替シンタックスは、処理後の値を返すタグにおいてのみ使用可能です：

- 4DTEXT
- 4DHTML
- 4DEVAL

(その他のタグ、たとえば 4DIF や 4DSCRIPT などでは、通常のシンタックスを使用しなければなりません)。

たとえば：

```
$4DEVAL(UserName)
```

は次の代わりになります：

```
<!--#4DEVAL(UserName)-->
```

このシンタックスの主な利点は、XML準拠のテンプレートが書けることです。一部の 4Dデベロッパーは、XML準拠のテンプレートを標準の XMLパーサーツールで作成・評価する必要があります。"<" 文字は XML属性値としては無効なため、ドキュメントのシンタックスを破らずに4Dタグの " <!-- --> " シンタックスを使用することはできませんでした。その一方で、"<" 文字をエスケープしてしまうと、4Dがタグを正常に解釈できなくなってしまいます。

たとえば、以下のコードは属性値の最初の "<" 文字のために XMLパースエラーを引き起こします：

```
<line x1="" y1=""/>
```

\$シンタックスを使用すると、パーサーによって以下のコードが評価されます：

```
<line x1="$4DEVAL($x)" y1="$4DEVAL($graphY1)"/>
```

ここで、\$4dtag と <--#4dtag--> は厳密には同じではないという点に注意が必要です。<--#4dtag--> とは異なり、\$4dtag は 4Dタグを繰り返し解釈 することはありません。\$ タグは常に一度だけ解釈され、その結果は標準テキストとして読まれます。

この違いの理由は、悪意あるコードの侵入を防ぐためにあります。悪意あるコードの侵入を防止 の章で説明されているように、ユーザーテキストを使用する場合に不要なタグの再解釈を避けるには、4DHTML タグではなく 4DTEXT タグの使用が強く推奨されます。4DTEXT を使用した場合、"<" などの特殊記号はエスケープされるため、<!--#4dtag expression--> シンタックスを使用している 4Dタグはすべて元の意味を失います。しかしながら、4DTEXT は \$ 記号をエスケープしないため、悪意あるコードの侵入を防ぐために \$4dtag (expression) シンタックスにおける再帰的処理はサポートしないことになりました。

以下の例では、使用されるシンタックスとタグによる処理の結果の違いを表しています：

```
// 例 1
myName:="" // 悪意あるコードの侵入
input:="My name is: "
PROCESS 4D TAGS(input;output)
// 4D は終了してまいります
```

```
// 例 2
myName:="<!--#4DHTML QUIT 4D-->" // 悪意あるコードの侵入
input:="My name is: <!--#4DTEXT myName-->"
PROCESS 4D TAGS(input;output)
// 結果は "My name is: <!--#4DHTML QUIT 4D-->"
```

```
// 例 3
myName:="$4DEVAL(QUIT 4D)" // 悪意あるコードの侵入
input:="My name is: <!--#4DTEXT myName-->"
PROCESS 4D TAGS(input;output)
// 結果は "My name is: $4DEVAL(QUIT 4D)"
```

\$4dtag シンタクスは、引用符や括弧の開閉ペアをサポートしているという点に注意してください。たとえば、以下の（非現実的な）文字列を評価しなければならない場合：

```
String(1) + "\"(hello)\""
```

以下のように書くことができます：

```
input:="$4DEVAL( String(1)+\"\\\"\"(hello)\\\"\\\"\")"
PROCESS 4D TAGS(input;output)
--> 結果は: 1"(hello)"
```

インターフェースページ

インターフェースページでは、プロジェクトインターフェースに関するさまざまな設定をおこないます。

一般

このエリアでは、表示に関する様々なオプションを設定します。

The screenshot shows the 'Formulas - Structure Settings' dialog box. The 'Interface' tab is selected. In the 'General' section, there is a dropdown for 'Font to use with the MESSAGE command' set to 'Default Font - 12 points'. Below it, under 'Display Windows', several checkboxes are checked: 'Splash screen', 'Flushing progress', and 'Printing progress'. There is also an unchecked checkbox for 'Use SDI mode on Windows'. A dropdown for 'Appearance' is set to 'Inherited'.

MESSAGEコマンドで使用されるフォント

選択... をクリックして、 MESSAGE コマンドで使用される文字のフォントとサイズを設定します。

4D が実行されているプラットフォームによって、デフォルトのフォントとサイズは異なります。

このプロパティは 4D の以下の部分にも影響します:

- エクスプローラーの特定のプレビューエリア
 - フォームエディターのルーラー
- 他のオプションでは、アプリケーションモードでのさまざまなウィンドウの表示を設定します。

- スプラッシュスクリーン: このオプションが選択解除されていると、アプリケーションモードにおいて、[カレントメニューバーのスプラッシュスクリーン](#)は表示されません。このウィンドウを非表示にした場合、ウィンドウの表示は、たとえば `On Startup` データベースメソッドによってプログラムで管理しなければなりません。
- フラッシュの進捗状況: このオプションがチェックされていると、キャッシュデータがフラッシュされる際、4D は画面左下にウィンドウを表示します。この処理はユーザー操作を一時的にブロックするため、ウィンドウを表示することでフラッシュがおこなわれていることをユーザーに通知することができます。

設定 > データベース > メモリ ページで [キャッシングフラッシュの周期](#) を設定できます。

- 印刷の進捗状況: 印刷時の印刷進捗状況を表示するダイアログを有効または無効にします。
- WindowsでSDIモードを使用する: このオプションが選択されていると、対応している環境で組み込みアプリが実行された場合に、4D は自動で SDIモード (Single-Document Interface) を有効にします。

このオプションは macOS でも有効にできますが、同プラットフォーム上で実行の際には無視されます。

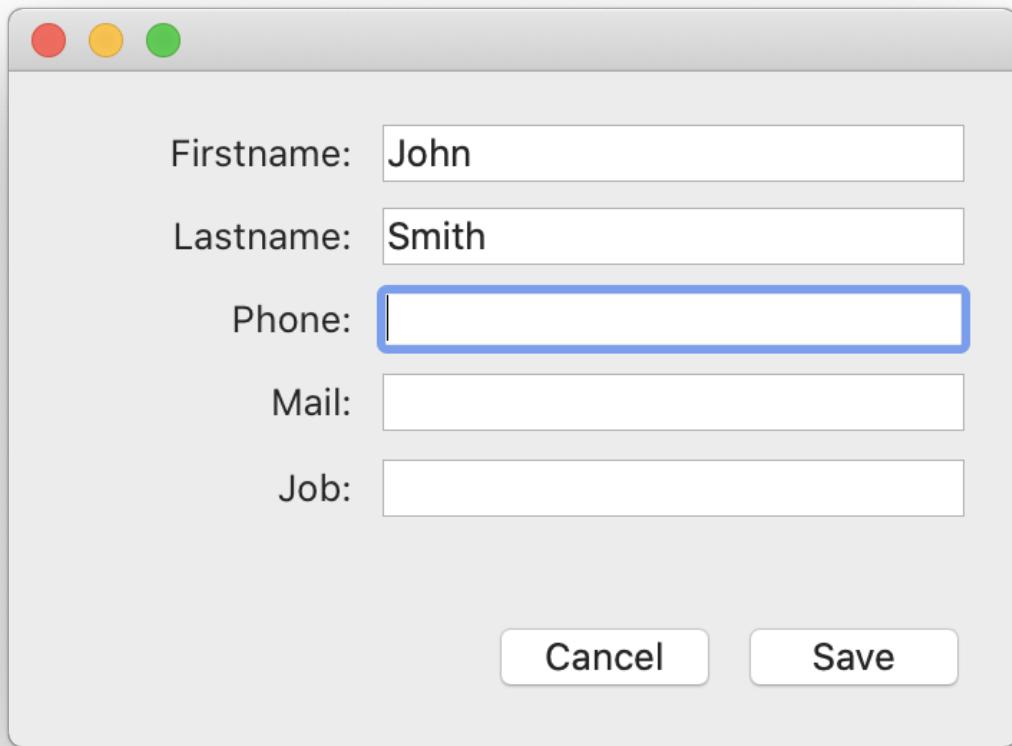
アピアランス

このメニューで、メインアプリケーションレベルにおいて使用するカラースキームを選択します。カラースキームは、フォーム内で使用されるテキスト、背景、ウィンドウなどのインターフェースカラーのグローバルなセットを定義します。

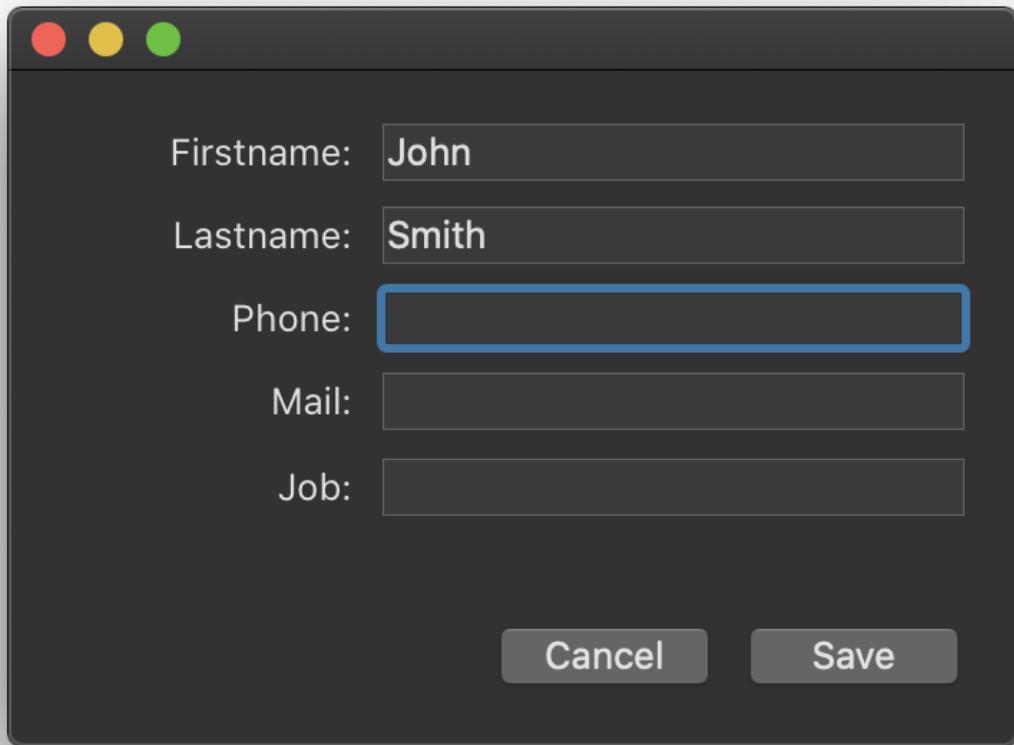
このオプションは macOS でのみ使用できます。Windows上では、"Light" テーマが常に使用されます。

以下のスキームが利用可能です:

- ライト: アプリケーションはデフォルトのライトテーマを使用します。



- ダーク: アプリケーションはデフォルトのダークテーマを使用します。



- 継承する (デフォルト): アプリケーションは次の優先レベル (例: OSユーザー設定) のテーマを継承します。

デフォルトのテーマは CSS で管理可能です。詳細については、[メディアエディタ](#) を参照してください。

メインアプリケーションスキームはデフォルトでフォームに適用されます。その一方で、それを以下の方法で上書きすることもできます:

- ワーキングセッションレベルでは [SET APPLICATION COLOR SCHEME](#) コマンドを使用できます。
- 各フォームレベル (最優先レベル) では、[カラースキーム](#) プロパティを使用できます。注記: 印刷時には、フォームは常に "ライト" スキームを使用します。

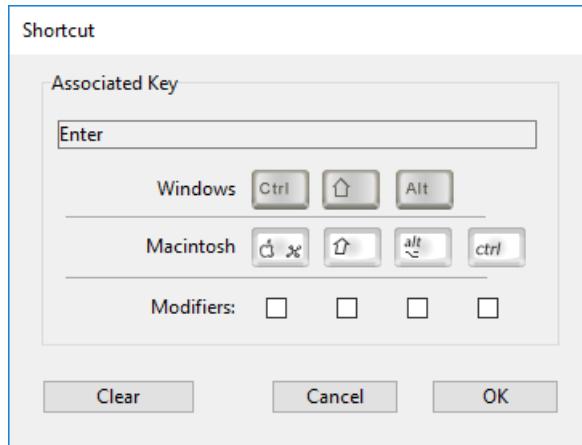
ショートカット

ショートカットエリアでは、4D のデスクトップアプリケーションにおける 3つの基本のフォーム操作に使うデフォルトショートカットを確認し、変更することができます。これらのショートカットは両プラットフォームで同じです。キーの形をしたアイコンは、Windows と macOS の対応するキーの組み合わせを示します。

デフォルトのショートカットは以下のとおりです:

- 入力フォーム確定: Enter
- 入力フォームキャンセル: Esc
- サブフォームに追加: Ctrl+Shift+/ (Windows) または Command+Shift+/ (macOS)

操作のショートカットを変更するには、対応する 編集 ボタンをクリックします。以下のダイアログボックスが表示されます:



ショートカットを変更するにはキーボードで新しいキーの組み合わせをタイプし、OK をクリックします。ショートカットを無効にするには、クリア ボタンをクリックします。

コンパイラーページ

これらのパラメーターは、[コンパイラー設定](#) で詳しく説明されています。

データベースページ

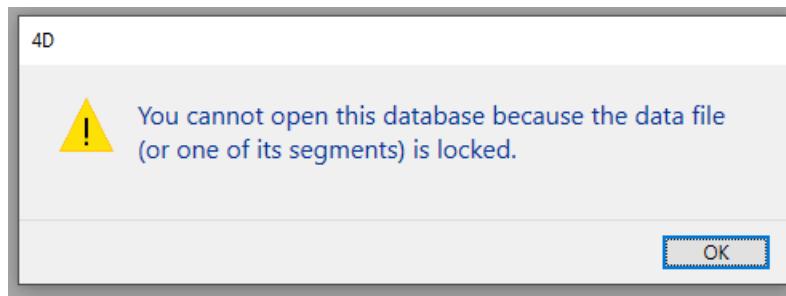
データストレージページ

このページでは、4Dデータベースが使用するデータストレージに関する設定をおこないます。

一般設定

読み込みのみのデータファイルの使用を許可する

このオプションを使用すると、OSレベルでロックされているデータファイルをアプリケーションから開けるようになります。4Dは、ロックされたデータファイルを開かないようにする自動的なメカニズムを実装しています。データファイルがロックされていると、4Dはデータベースを開かず、以下のような警告メッセージを表示します：



このオプションを選択しないかぎり、ロックされたデータファイルを開くことはできません（4Dデータベースのデフォルト動作）。

ロックされたファイルについて

ロックされたファイルは読むことはできても内容を変更することはできません。たとえば（DVDのような）編集不可のメディアに格納されたファイルや、このようなメディアからコピーされたファイルは、ロックされていることがあります。DVDに格納されたプロジェクトを使用するような場合、ロックされたデータファイルを使用できるようアプリケーションを設定することができます。しかしこの場合、データの追加・編集が保存されないロックされているデータファイルを不注意に使用してしまうリスクがあります。このような状況にならないよう、4Dではデフォルトでロックされたデータファイルを開くことを禁止しています。

テンポラリーフォルダーの場所

このエリアでは、4D実行中に作成されるテンポラリーファイルの場所を設定できます。テンポラリーファイルのフォルダーは、一時的にメモリ中のデータをディスクに保存するために必要に応じてアプリケーションが使用します。

現在のフォルダーの場所は "現在：" エリアに表示されます。このエリアをクリックするとパス名がスクロールダウンリスト形式で表示されます：



3つのオプションから選択できます：

- システム：このオプションが選択されると、4DのテンポラリーファイルはWindowsまたはmacOSが指定する場所に配置されたフォルダーに作成されます。システムが指定する場所は `Temporary folder` コマンドで知ることができます。ファイルは、データベース名とユニークな識別子からなるサブフォルダーに置かれます。
- データファイルフォルダー（デフォルト）：このオプションが選択されると、4Dのテンポラリーファイルはデータベースのデータファイルと同階層に配置される "temporary files" フォルダーに作成されます。
- ユーザー指定：このオプションは、場所をカスタマイズするのに使用します。場所のオプションを変更した場合、新しい設定を反映するにはデータベースを再起動する必要があります。4Dは選択されたフォルダーに書き込みアクセス権があるかを確認します。アクセス権がなければ、使用できるフォルダーが見つかるまで4Dは他のオプションを試します。

このオプションは、ストラクチャー定義がXML形式で書き出されたとき、ストラクチャーの "extra properties" に格納されます（[ストラクチャー](#)）

定義の書き出しと読み込み 参照)。

テキスト比較

これらのオプションのいずれかを変更した場合、新しい設定を反映するにはアプリケーションを終了し、再起動しなければなりません。データベースが再び開かれると、すべてのインデックスが自動で再作成されます。

- 文字列の途中に含まれる@はワイルドカードとして扱わない: クエリ条件や文字列比較の際に "@" をどのように解釈するかを設定します。このオプションが選択されていない場合 (デフォルト設定)、"@" はワイルドカードとして扱われ、あらゆる文字の代わりとみなされます ([ワイルドカード記号 \(@\) 参照](#))。

このオプションが選択されている場合、単語内にある "@" は普通の文字として扱われます。この設定は、"@" が文字列内で使用される電子メールアドレスの検索などに有効です。このオプションは検索、並び替え、文字列比較、およびテーブルに格納されるデータや配列などメモリ中のデータに影響を与えます。(インデックス付きか否かにかかわらず) 文字型やテキスト型のフィールドおよび変数が、検索や並び替え時に "@" 文字がどのように解釈されるかの影響を受けます。

注:

- 検索において、"@" が検索条件の先頭か最後にある場合、"@" はワイルドカードとして扱われます。単語の中に "@" 文字がある場合のみ (例: bill@cgi.com)、4D は異なった扱いをします。
 - このオプションは object引数に "@" ワイルドキャラクターを受け入れる [オブジェクト\(フォーム\)](#) テーマのコマンドの動作にも影響を与えます。
 - セキュリティの理由で、データベースの Administrator または Designer のみがこのパラメーターを変更できます。
- 現在のデータ言語: 文字列の処理と比較に使用する言語を設定します。言語の選択は、テキストの並べ替えや検索、文字の大小などの比較ルール等に直接影響を与えます。ただし、テキストの翻訳や日付・時刻・通貨のフォーマットはシステムの言語設定が使用され、この設定には影響されません。デフォルトで 4D はシステム言語を使用します。
- つまり、4Dプロジェクトはシステム言語とは異なる言語で動作することができます。プロジェクトが開かれると、4Dエンジンはデータファイルに使用されている言語を検知し、(インターフィーリーやコンパイルモードの) ランゲージに提供します。データベースエンジン、あるいはランゲージのいずれがテキスト比較をおこなうかに関わらず、同じ言語が使用されます。

4D環境設定でも言語を設定できます ([一般ページ 参照](#))。この場合、その設定は新規に作成されるデータベースに適用されます。

- 非文字・非数字のみをキーワード区切り文字とする: 4D が使用するキーワード区切り文字の設定を変更し、その結果、作成されるキーワードインデックスに影響を与えます。このオプションが選択されていない場合、4D は言語上の特質を考慮する洗練されたアルゴリズムを使用します。

このアルゴリズムは、文字列中をダブルクリックしたときに選択範囲を決定するためワープロソフトが使用するものと同じです。このアルゴリズムに関する詳細は以下の Webページを参照ください: <http://userguide.icu-project.org/boundaryanalysis>。

このオプションが選択されている場合、4D は簡易的なアルゴリズムを使用します。この設定では、文字でも数字でもない文字がキーワード区切り文字として扱われます。この設定は日本語など特定の言語の要求に沿うものです。

- テキスト検索用の文字列比較を使用する: このオプションは日本語が選択されている場合にのみ表示されます。このオプションは "カタカナ-ひらがなでの長音記号" および "ゝ" または "ゞ" などの繰り返し記号 (踊り字) などの文字の解釈を変更します。一般的には、この設定が有効化されている方が日本語話者にとって望ましい結果が得られます。

MeCab のサポート (日本語版)

日本語版の 4D では、MeCab ライブラーがサポートされており、日本語用に調整されたキーワードのインデックスアルゴリズムを兼ね備えています。

日本語版の 4D ではこのアルゴリズムがデフォルトで使用されています。必要であれば MeCab アルゴリズムを無効にして、以前と同じ ICU ライブラーを使用することもできます。

MeCab を無効化するには、非文字・非数字のみをキーワード区切り文字とする のオプションにチェックを入れます:

Current data language: **Japanese**

Consider only non-alphanumeric chars for keywords

メモリページ

このページでは、データベースのキャッシュメモリに関する設定をおこないます。

データベースキャッシュ設定

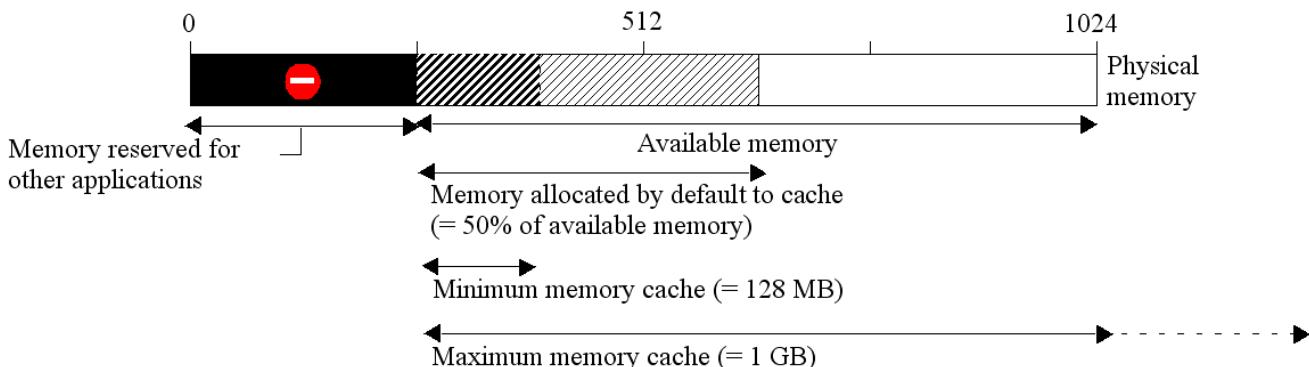
- 起動時にキャッシュサイズを計算: このオプションが選択されている場合、キャッシュメモリの管理は、設定された制限に基づき、4D起動時にキャッシュサイズが計算されます。これにより、ほとんどのケースで高パフォーマンスのメモリ設定がおこなわれます。キャッシュメモリのサイズは設定されたパラメーターに基づき動的に計算されます。デフォルトで提案される値は標準の 4D の使用状況に対応します。
 - 他のアプリケーションとシステムのために予約するメモリ: システムや他のアプリケーションが使用するために取り置く RAM メモリ量。4D が実行されるマシン上で他のアプリケーションも実行する場合、必要に応じてこの値を増やします。
 - 利用可能なメモリからキャッシュに使用する率: 残りのメモリからキャッシュに割り当てる量の率。デフォルトでキャッシュに割り当てるサイズを取得するためには、以下の計算式を適用します: (物理メモリ - 予約したメモリ) X キャッシュに使用するメモリのパーセンテージ。動的に計算するモードの場合、キャッシュメモリのサイズはアプリケーションやシステムのニーズに応じて動的に変化します。以下のオプションを使用して上限と下限を設定できます:
 - 最小サイズ: キャッシュ用に予約するメモリの最小量。この値は 100MB 以上でなければなりません。
 - 最大サイズ: キャッシュが使用することのできるメモリの最大量。この値は実質的には無制限です。¥ 制限の設定は、メモリ搭載量が不明であるマシン用にアプリケーションを配布する際に便利です。この場合、制限を設定することで最低限のパフォーマンスを保証できます。この動作を図示すると以下のようにになります:

キャッシュメモリの計算例:

予約するメモリ量 = 256 MB

キャッシュに利用するパーセンテージ = 50%

最大サイズ = 1 GB 最小サイズ = 128 MB



- 起動時にキャッシュサイズを計算オプションがチェックされていない場合: このモードでは、データベースのキャッシュメモリサイズを開発者が決定します。4D はキャッシュメモリを設定する入力エリアと、物理メモリに関する情報（マシンに実装されたRAM）、現在のキャッシュ、そして再起動後のキャッシュサイズを表示します。

入力されたキャッシュサイズは、マシンのリソース状況にかかわらず、4Dデータベース用に予約されます。この設定は、メモリ搭載量が分かれているなど、特定の状況で使用できます。（ほとんどのケースで起動時計算モードのキャッシュで良いパフォーマンスが提供されます）。

- キャッシュをディスクに保存 ... 秒/分: キャッシュ中のデータを自動的に保存する間隔を設定します。4D はキャッシュ中のデータを定期的に保存します。この間隔を 1 秒から 500 分の範囲で設定できます。デフォルトの設定値は 20 秒です。この保存は、キャッシュがいっぱいになった場合や、アプリケーションを終了する際にもおこなわれます。また **FLUSH CACHE** コマンドを使って、いつでもフラッシュをトリガーできます。

大量のデータ入力が予期される場合は、この間隔を短くすることを検討してください。停電などの理由でマシンが停止すると、前回の保存以降に入力されたデータが失われてしまいます（データログファイルをとっていれば復旧可能です）。

キャッシュがフラッシュされるたびにデータベースの動作が遅くなる場合、周期を調整する必要があります。動作が遅くなるのは、大量のレコードがディスクにフラッシュされるためです。フラッシュ周期を短くすることで、各フラッシュ時に保存されるレコード数を減らすことができ、動作も速くなります。

デフォルトで 4D はキャッシュがフラッシュされていることを示す小さなウィンドウを表示します。このウィンドウを表示したくない場合、[インターフェースページ](#) の フラッシュの進捗状況 オプションの選択を解除します。

クライアント-サーバーページ

クライアント/サーバーページには、クライアント/サーバーモードでデータベースを使用する際に使用されるパラメーターが集められています。これらの設定は、リモートモードでデータベースが使用されるときにのみ使用されます。

ネットワークオプションページ

ネットワーク

起動時にデータベースを公開する

このオプションを使用して、起動された 4D Server データベースが公開データベースのリストに表示されるかどうかを指定できます。

- このオプションが選択されていると（デフォルト）、データベースは公開され、公開データベースの一覧に表示されます（リモート4D の 利用可能タブ）。
- このオプションがチェックされていないと、データベースは公開されず、公開データベースの一覧に表示されません。接続するには、接続ダイアログボックスの カスタム タブにデータベースのアドレスを手入力しなければなりません。

この設定を変更した場合、変更を反映するためサーバーデータベースを再起動する必要があります。

公開名

このオプションでは、4D Server データベースの公開名を変更できます。この名前は接続ダイアログボックスの 利用可能 ページに表示されます（[4D Serverデータベースへの接続 参照](#)）。デフォルトで 4D Server はプロジェクトファイル名を使用します。これを好きな名前に変更できます。

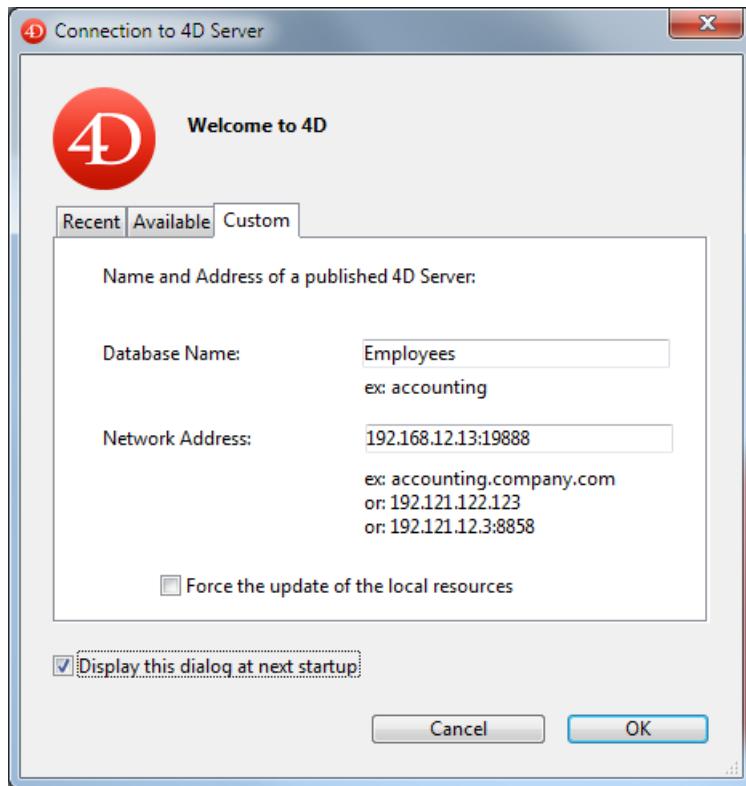
このパラメーターはカスタムのクライアント-サーバーアプリケーションでは使用されません。クライアントアプリケーションは接続ダイアログを経由せずにサーバーに直接接続します。しかしエラーが発生すると、このダイアログが表示されます。この場合、サーバーアプリケーションの公開名はコンパイルされたプロジェクトの名前です。

ポート番号

このオプションで、4D Server がデータベースを公開する TCPポート番号を変更できます。この情報は、プロジェクト及び各クライアントマシンに格納されます。4D Server とリモートモードの 4D が使用するデフォルトの TCPポート番号は 19813 です。

TCPプロトコルを使用して、1台のマシン上で複数の 4D アプリケーションを同時に使用したい場合にこの値の変更が必要です。この場合、アプリケーションごとに異なるポート番号を割り当てなければなりません。4D Server または 4D からこの値を変更すると、データベースに接続しているすべての 4D マシンに変更が通知されます。

接続していないクライアントを更新するには、次回の接続時に接続ダイアログボックスの カスタム ページにて、サーバーマシンの IPアドレスに続けてコロン、そして新しいポート番号を入力します。たとえば、新しいポート番号が 19888 あるとき：



4Dクライアントと同じポート番号で公開されているデータベースだけが、接続ダイアログの利用可能ページに表示されます。

4D Server とポート番号

4D Server は 3つの TCPポートを使用して、内部サーバーとクライアントの通信をおこないます：

- SQLサーバー：デフォルトで 19812（設定の "SQL" ページで変更可）。
- アプリケーションサーバー：デフォルトで 19813（設定の "クライアント-サーバー" ページで変更可）。
- DB4Dサーバー（データベースサーバー）：デフォルトで 19814。このポート番号は直接変更できませんが、常にアプリケーションサーバのポート番号 +1 です。

4Dクライアントが 4D Server に接続するとき、アプリケーションサーバのポート番号（19813 または接続ダイアログボックスの IPアドレス欄でコロンの後ろに指定された番号）を使用して接続します。その後の、それぞれ対応するポートを介した他のサーバーへの接続は自動です。再度ポート番号を指定する必要はありません。

ルーターやファイアウォール越しに接続する場合には、この 3つのポートを明示的に開く必要があります。

ドメインサーバーによるユーザーの認証

このオプションは Windows上の 4D Server データベースにおいて SSO（Single Sign On）機能の実装を可能にします。このオプションを有効にすると、4D はバックグラウンドで Windows ドメインサーバーの Active Directory に接続し、提供されている認証トークンを取得します。このオプションの詳細については [Windowsでのシングルサインオン\(SSO\)](#) を参照ください。

サービスプリンシパル名 (SPN)

Single Sign On (SSO) が有効になっている場合（上述参照）、認証プロトコルにケルベロスを使用するには、このフィールドを設定する必要があります。このオプションの詳細については [Windowsでのシングルサインオン\(SSO\)](#) を参照ください。

クライアント/サーバー接続タイムアウト

このサーモメーターで、4D Server とクライアントマシン間の（一定時間活動がないときに接続を閉じる）タイムアウトを設定できます。無制限オプションは、タイムアウトを設定しないことを意味します。このオプションが選択されると、クライアントのアクティビティコントロールはおこなわれません。

タイムアウト時間が選択されると、その間にリクエストを受信しなかった場合、サーバーはそのクライアントとの接続を閉じます。

クライアント-サーバー通信

Execute On Clientのために起動時にクライアント登録

このオプションが選択されていると、データベースに接続するすべての 4Dリモートマシン上でメソッドをリモート実行できます。このメカニズムについては [クライアントマシン上のストアドプロシージャ](#) で説明しています。

クライアント-サーバー通信の暗号化

このオプションを使用して、サーバーマシンと 4Dリモートマシン間通信の保護モードを有効にできます。このオプションについては [クライアント/サーバー接続の暗号化](#) で説明しています。

セッション中に "Resources" フォルダーを更新

この設定は、データベースの Resources フォルダーがセッション中に更新された場合について、接続中のクライアントマシンにおける同フォルダーのローカルインスタンスの更新モードを包括的に指定します (Resources フォルダーは、セッションが開かれたたびにリモートマシン上で自動的に同期されます)。3つの選択肢があります：

- しない: ローカルの Resources フォルダーはセッション中に更新されません。サーバーから送信される通知は無視されます。ローカルリソースを更新アクションメニュー命令 ([リソースエクスプローラーを使用する](#) 参照) を使用すれば、ローカルの Resources フォルダーを手動で更新することができます。
- 常に: セッション中にサーバーから通知が送信されると、ローカルの Resources フォルダーは自動で同期されます。
- その都度指定: サーバーから通知を受け取ると、クライアントマシン上でダイアログボックスが表示されます。ユーザーはローカルの Resources フォルダーの同期を受け入れ、あるいは拒否できます。

Resources フォルダーは、データベースインターフェースで使用されるカスタムファイルを格納しています (翻訳ファイルやピクチャーなど)。このフォルダーの内容が更新されたときには、自動又は手動メカニズムを使用して各クライアントに通知できます。詳細については、[リソースフォルダの管理](#) を参照ください。

IP設定ページ

許可-拒否設定表

この表を使用して、4Dリモートマシンの IPアドレスに基づき、データベースへのアクセスコントロールルールを設定できます。このオプションを使用して、たとえば戦略アプリケーションなどのセキュリティを高めることができます。

Web接続は、この設定表でコントロールされません。

設定表の動作は以下のとおりです：

- "許可-拒否" 列では、ポップアップメニューを使用して適用するルールを選択します (許可または拒否)。ルールを追加するには、追加ボタンをクリックします。すると、新しい行が表に追加されます。削除 ボタンで選択した行を削除できます。
- "IPアドレス" 列で、ルールに関連する IPアドレスを指定します。アドレスを指定するには、選択した行のセルをクリックし、以下の形式でアドレスを入力します: 123.45.67.89 (IPv4) または 2001:0DB8:0000:85A3:0000:0000:AC1F:8001 (IPv6)。* (アスタリスク) 文字をアドレスの末尾に使用して、範囲を指定することもできます。たとえば、192.168.* は 192.168 で始まるすべてのアドレスを示します。
- ルールの適用は、表中の表示順に基づきます。2つのルールが矛盾する設定の場合、より上に設定されているルールが優先されます。行の順番を変更するには、列のヘッダーをクリックしてソートをおこなったり、ドラッグ & ドロップで移動したりすることができます。ドラッグ & ドロップで移動したりすることができます。
- セキュリティのため、ルールにより明示的に許可されたアドレスのみが接続を許可されます。言い換えれば、表に拒否ルールしか定義されていない場合、許可ルールに適合するアドレスがないため、すべてのアドレスからの接続が拒否されます。特定のアドレスからの接続のみを拒否したい場合 (そして他を許可したい場合)、許可 * ルールを表の最後に追加します。たとえば:
 - 拒否 192.168.* (192.168 で始まるアドレスを拒否)
 - 許可 * (他のアドレスはすべて許可)

デフォルトでは、4D Server にアクセス制限はありません。最初の行には * (すべてのアドレス) に対する許可ルールが設定されています。

Web ページ

Web ページのタブを使用して、4D に統合された Web サーバーの様々な設定（セキュリティ、開始オプション、接続、Web サービス等）にアクセスできます。4D Web サーバーの動作に関する詳細は [Web サーバー](#) を参照ください。また、Web サービスに関する詳細は [Web サービスの公開と使用](#) を参照ください。

設定

公開情報

開始時に Web サーバーを起動

4D アプリケーションの起動時に Web サーバーを開始するか指定します。このオプションは [4D Web サーバーの開始](#) で説明しています。

HTTP を有効化

安全でない接続を Web サーバーが受け入れるかどうかを示します。[HTTP を有効化](#) 参照。

HTTP ポート

HTTP 接続を受け付ける IP (TCP) ポート番号。[HTTP ポート](#) 参照。

IP アドレス

4D Web サーバーが HTTP リクエストを受け付ける IP アドレスを指定できます（4D ローカルおよび 4D Server）。[リクエストを受け付ける IP アドレス](#) 参照

HTTPS を有効にする

安全な接続を Web サーバーが受け入れるかどうかを示します。[HTTPS を有効にする](#) 参照。

HTTPS ポート

TLS (HTTPS プロトコル) を使用したセキュアな HTTP 接続に対して Web サーバーが使用する TCP/IP ポート番号を指定できます。[HTTPS ポート](#) 参照

"4DSYNC" URL を使用したデータベースアクセスを許可

互換性に関する注記：このオプションは [廃止予定](#) です。今後、HTTP を介したデータベースアクセスには ORDA のリモートデータストア機能と RESTful クエストの使用が推奨されます。

パス

デフォルト HTML ルート

Web サイトファイルのデフォルトの位置を指定し、それより上のファイルにはアクセス不能なディスク上の階層レベルを指定します。[ルートフォルダー](#) 参照。

デフォルトホームページ

Web サーバー用のデフォルトホームページを指定します。[デフォルトホームページ](#) 参照。

オプション (I)

キヤッショウ

4D Webキャッシングを使用する

Webページキャッシングを有効化します。 [キャッシング 参照。](#)

ページキャッシングサイズ

キャッシングサイズを指定します。 [キャッシング 参照。](#)

キャッシングクリア

いつでもページやイメージをキャッシングからクリアできます（たとえば、スタイルページを更新し、キャッシングにそれをリロードさせたい場合）。これをおこなうには、キャッシングクリア ボタンをクリックします。キャッシングは即座にクリアされます。

特殊なURL </4DCACHECLEAR> を使用することもできます。

Webプロセス

このエリアでは、ユーザーセッションとそれに関連するプロセスを Webサーバーがどのように管理するかを設定します。

旧式セッション オプションは、4D v18 R6 以前のバージョンで作成されたデータベース/プロジェクトにおいて互換性のためにのみ利用可能です。

スケーラブルセッション（マルチプロセスセッション）

このオプションを選択すると（推奨）、ユーザーセッションは Session オブジェクトを介して管理されます。 [ユーザーセッション](#) のページを参照ください。

セッションなし

このオプションが選択されている場合、Webサーバーは [ユーザーセッション](#) 専用のサポートを提供しません。 Webクライアントからの連続したリクエストはどちらも常に独立しており、サーバー上でコンテキストは維持されません。

このモードでは、以下の追加の Web サーバー設定を設定することができます：

- [最大同時Webプロセス](#)
- [一時的なコンテキストを再利用する（リモートモード）](#)
- [プリエンプティブプロセスを使用](#)

旧式セッション（シングルプロセスセッション）

互換性に関する注記： このオプションは 4D v18 R6 以前のバージョンで作成されたデータベース/プロジェクトでのみ利用可能です。

このオプションは 4D HTTP サーバーの旧式ユーザーセッションの管理を有効化します。この機構は [Webセッション管理（旧式）](#) にて詳細に説明されています。 [旧式セッション（自動セッション管理）](#) 参照。

このオプションが選択されていると、[一時的なコンテキストを再利用する（リモートモード）](#) オプションも自動で選択され、ロックされます。

最大同時Webプロセス

[スケーラブルセッションモード](#) の場合には利用できません。

Webプロセスの最大同時接続数の厳格な上限です。 [最大同時Webプロセス](#) 参照。

一時的なコンテキストを再利用する

[スケーラブルセッションモード](#) の場合には利用できません。

リモートモードで実行されている 4D Webサーバーの動作を最適化できます。 [一時的なコンテキストを再利用する（リモートモード）](#) 参照。

プリエンプティブプロセスを使用

[スケーラブルセッションモード](#) の場合には利用できません。

コンパイル済みアプリケーションにおいてプリエンプティブWebプロセスを有効化します。プリエンプティブプロセスを使用が選択されているとき、Web関連のコード (4D グと Webデータベースメソッドを含む) は、コンパイル時にプリエンティブな実行が可能かどうかが評価されます。詳細な情報については、[プリエンプティブWebプロセスの使用](#) を参照ください。

このオプションは スケーラブルセッション、RESTプロセス (コンパイル済みモード)、および Webサービスプロセス (サーバーあるいはクライアント) には適用されません。[Webサーバーにおいてプリエンプティブモードを有効化する](#) 参照。

非動作プロセスのタイムアウト

[スケーラブルセッションモード](#) の場合には利用できません。

サーバー上で活動していない Webプロセスを閉じるための最大タイムアウト時間を設定できます。[非アクティブセッションタイムアウト](#) 参照。

Webパスワード

Webサーバーに対して適用する認証システムを設定します。3つのオプションから選択できます:

カスタムの認証 (デフォルト)

BASIC認証のパスワード

DIGEST認証のパスワード

カスタムの認証 を使用することが推奨されています。Web 開発 ドキュメンテーションの[認証](#) の章を参照ください。

オプション (II)

テキスト変換

拡張文字をそのまま送信

[廃止予定の設定](#) 参照。

文字コード

4D Webサーバーが使用する文字セットを定義します。[文字コード](#) 参照。

Keep-Alive接続を使用する

[廃止予定の設定](#) 参照。

CORS設定

CORSを有効化

クロスオリジンリソースシェアリング (CORS) サービスを有効化します。[CORSを有効化](#) 参照。

ドメイン名/許可されたHTTPメソッド

CORSサービスで許可されたホストとメソッドの一覧。[CORS設定](#) 参照。

ログ (タイプ)

ログフォーマット

4D Web サーバーが受け取るリクエストのログを開始/停止します。ログは、*logweb.txt* ファイルに記録され、そのフォーマットを指定することができます。[ログの記録](#) 参照。

リクエストのログファイルの有効/無効は [WEB SET OPTION](#) コマンドを使用したプログラミングでも切り替えられます。

ログフォーマットメニューでは、次のオプションを提供します。

- ログファイルなし: このオプションが選択されると、4D はリクエストのログファイルを作成しません。
- CLF (Common Log Format): このオプションが選択されると、リクエストのログが CLFフォーマットで作成されます。CLFフォーマットでは、それぞれのリクエストが行単位でファイル内に表示されます:
host rfc931 user [DD/MMM/YYYY:HH:MM:SS] "request" state length
各フィールドはスペースによって区切られ、各行は CR/LF シーケンス (character 13, character 10) で終わります。
 - host: クライアントの IPアドレス (例: 192.100.100.10)
 - rfc931: 4Dによって作成されない情報。常に - (マイナス記号) です。
 - use: 認証されているユーザー名、あるいは、- (マイナス記号)。ユーザー名にスペースが含まれると、_ (下線) に置き換わります。
 - DD: 日、MMM: 月を表す3文字の略号 (Jan, Febなど)、YYYY: 年、HH: 時間、MM: 分、SS: 秒。

日付と時間はサーバーのローカルタイム。

- request: クライアントによって送られたリクエスト (例: GET /index.htm HTTP/1.0)
- state: サーバーの返答。
- length: 返されたデータ (HTTPヘッダー以外) のサイズまたは 0。

注: パフォーマンス上の理由により、操作はメモリのバッファーに 1Kbのパケットで保存されてから、ディスクに書き込まれます。5秒間リクエストが発生しなくても、操作はディスクに書き込まれます。state として取り得る値は以下の通り:

200: OK
204: No contents
302: Redirection
304: Not modified
400: Incorrect request
401: Authentication required
404: Not found
500: Internal error
CLFフォーマットはカスタマイズされません。

- DLF (Combined Log Format): このオプションが選択されると、リクエストログが DLFフォーマットで作成されます。DLFフォーマットは CLFフォーマットと類似していて、まったく同じ構造を使用します。さらに、各リクエストの最後に2つのHTTPフィールド、Referer と User-agent を追加します。
 - Referer: リクエストされたドキュメントを指しているページの URL を含みます。
 - User-agent: リクエストのオリジンにおけるクライアントのブラウザーまたはソフトウェアの名前とバージョンを含みます。

DLFフォーマットはカスタマイズされません。

- ELF (Extended Log Format): このオプションが選択されると、リクエストログが ELFフォーマットで作成されます。ELFフォーマットは HTTPユーザー界で広く普及しています。そして、特別なニーズに応える洗練されたログを構築します。この理由により、ELFフォーマットはカスタマイズされます。記録するフィールドやそのフィールドをファイルへ挿入する順番を選択することができます。
- WLF (WebStar Log Format): このオプションが選択されると、リクエストログが WLFフォーマットで作成されます。WLFフォーマットは 4D WebSTAR サーバー用として特別に開発されました。いくつかの追加フィールドを持つ以外、EFLフォーマットと似ています。EFLフォーマットと同様、カスタマイズされます。

フィールドの設定

ELF (Extended Log Format) または WLF (WebStar Log Format) を選択すると、選択されたフォーマットに対して利用可能なフィールドが表示されます。ログに含む各フィールドを選択する必要があります。ログに含む各フィールドを選択する必要があります。

注: 同じフィールドを 2度選択することはできません。

各フォーマットで利用可能なフィールド (アルファベット順) とその内容を以下のテーブルに示します:

フィールド	ELF	WLF	値
BYTES_RECEIVED		○	サーバーが受け取ったバイト数
BYTES_SENT	○	○	サーバーがクライアントに送ったバイト数
C_DNS	○	○	DNSのIPアドレス (ELF: C_IP フィールドと同一のフィールド)
C_IP	○	○	クライアントの IPアドレス (例: 192.100.100.10)
CONNECTION_ID		○	接続ID番号
CS(COOKIE)	○	○	HTTPリクエストに格納されている cookies に関する情報
CS(HOST)	○	○	HTTPリクエストの Hostフィールド
CS(REFERER)	○	○	リクエストされたドキュメントを指すページの URL
CS(USER_AGENT)	○	○	ソフトウェアとクライアントのオペレーティングシステムに関する情報
CS_SIP	○	○	サーバーの IPアドレス
CS_URI	○	○	リクエストが作成された URI
CS_URI_QUERY	○	○	リクエストのクエリ引数
CS_URI_STEM	○	○	クエリ引数のないリクエストのパート
DATE	○	○	DD: 日、MMM: 月を表す3文字の略号 (Jan, Febなど)、YYYY: 年
METHOD	○	○	サーバーへ送られたリクエスト用の HTTPメソッド
PATH_ARGS		○	CGI引数: "\$" の後に続く文字列
STATUS	○	○	サーバーの返答
TIME	○	○	HH: 時間、MM: 分、SS: 秒
TRANSFER_TIME	○	○	返答を作成するためにサーバーが要求した時間
USER	○	○	認証されているユーザー名、あるいは、- (マイナス記号)。
			ユーザー名にスペースが含まれると、_ (下線) に置き換わります。
URL		○	クライアントがリクエストした URL

日付と時間は GMTで表されます。

ログ (バックアップ)

リクエストログの自動バックアップ設定を指定します。最初に、頻度 (日、週などの単位) またはファイルサイズの上限に対応するラジオボタンをクリックして選択します。必要に応じて、バックアップする正確な時間を指定します。

- バックアップしない: 周期的なバックアップ機能が無効になっています。
- X 時間ごと: 1時間単位でバックアップをプログラムする際、このオプションを使用します。1 から 24 の値を入力します。
 - 開始時刻: 最初のバックアップ開始時間の設定に使用します。
- X 日ごと: 1日単位でバックアップをプログラムする際、このオプションを使用します。バックアップを毎日実行するには、1を入力します。このオプションをチェックすると、バックアップの開始時間を指定しなければなりません。
- X 週ごと: 1週間単位でバックアップをプログラムする際、このオプションを使用します。たとえば、毎週バックアップをおこなうには 1 と設定します。たとえば、毎週バックアップをおこなうには 1 と設定します。このオプションをチェックすると、バックアップを開始する曜日と時間を指定しなければなりません。複数の曜日を選択することもできます。
- X 月ごと: 1ヶ月単位でバックアップをプログラムする際、このオプションを使用します。たとえば、毎月バックアップをおこなうには 1 と設定します。たとえば、毎月バックアップをおこなうには 1 と設定します。
- X MB (サイズ指定): カレントのリクエストログのファイルサイズに基づいてバックアップをプログラムする際、このオプションを使用します。ファイルが指定サイズに達すると、バックアップが自動的に起動します。サイズ制限は 1、10、100 または 1000MB ごとに設定可能です。

スケジュールされたバックアップの場合で、バックアップが開始する予定となっているときに Webサーバーが起動していないと、次回の起動において 4D はバックアップが失敗したと見なし、データベース設定で示されている適切な設定を適用します。

Webサービス

このページのオプションを使用して 4Dプロジェクトの Webサービスを有効にし、(サーバーおよびクライアントの) 設定をすることができます。

4Dにおける Webサービスサポートについては [Web サービスの公開と使用](#) を参照ください。

サーバー側設定

このエリアでは 4D を Webサービスサーバーとして使用する (つまり、Webサービスの形でプロジェクトメソッドを公開する) ための設定をおこないます。

- Webサービスリクエストを許可する: このオプションを使用して Webサービスクライアントからのリクエストを受け付けるかどうかを設定します。このオプションが選択されていない場合、4D は SOAPリクエストを拒否し、メソッドがWSDLを公開 属性を設定されていても WSDL を生成しません。このオプションを選択すると、4D はメソッドプロパティに基づき WSDLファイルを生成します。
- Webサービス名: このエリアでは Webサービスの "包括的な名称" を変更できます。この名前は、SOAPサーバーレベルでサービスを識別するために使用されます。デフォルトで 4D は "A_WebService" を使用します。
- Webサービス名前空間: このエリアは 4D が公開する Webサービスの名前空間を設定するのに使用します。インターネットに公開される各 Webサービスはユニークでなければなりません。Webサービス名をユニークにするために XML名前空間が使用されます。名前空間は任意の文字列で、XMLタグをユニークに識別するために使用されます。典型的には、名前空間は会社の URL で始まります (<http://mycompany.com/mynamespace>)。指定された文字列がユニークである限り、指定した URL に何か付け加える必要はありません。デフォルトで 4D は以下の名前空間を使用します: <http://www.4d.com/namespace/default>。

タグ名に関する XML標準に準拠するために、使用する文字には空白が含まれていてはなりません。また数字から始まっているではありません。非互換性を避けるために、アクセント文字や日本語などの拡張文字は使用しないことを推奨します。

クライアント側設定

このエリアでは Webサービスクライアントとして 4D を使用する (つまり、ネットワーク上に公開されているサービスにサブスクライブする) ための設定をおこないます。

- ウィザードメソッドプリフィックス: このエリアでは、Webサービスウィザードを使用して 4D が自動生成するプロキシメソッドの接頭辞を設定します。プロキシプロジェクトメソッドは 4Dアプリケーションと Webサービスサーバーとのリンクを形成します。デフォルトで "proxy_" が使用されます。

Web機能

このページには、RESTサーバーなどの高度な Web機能を有効化および制御するためのオプションが含まれています。

公開

RESTサーバーとして公開

RESTサーバーを開始/停止します。 [RESTサーバー設定](#) 参照。

アクセス

この設定は、RESTリクエストを使って 4Dデータベースへのリンクを設立することのできる 4Dユーザーのグループを指定します。 [アクセス権の設定](#) 参照。

Web Studio

Web Studio へのアクセスを有効化する

Web Studio へのアクセスを有効化します。さらに、プロジェクトレベルごとの設定が必要です。

SQL ページ

このページでは [4D SQLサーバー](#) の公開パラメーターやアクセス権、および 4D SQLエンジンの動作に関する設定をおこないます。

SQLサーバー公開

doc.4d.com の [4D SQLサーバの設定](#) を参照ください。

デフォルトスキーマ用のSQLサーバーアクセス権

doc.4d.com の [4D SQLサーバの設定](#) を参照ください。

SQLエンジンオプション

doc.4d.com の [SQLエンジンオプション](#) を参照ください。

PHP ページ

このページで PHP 実行に関する設定をおこなうことで、4D から直接 PHP スクリプトを実行することができます (4D ランゲージリファレンス マニュアルの [4D で PHP スクリプトを実行する](#) を参照ください)。

インタープリター

- IP アドレスとポート番号

4D はデフォルトで使用される FastCGI PHP インタープリターを提供しています。内部的なアーキテクチャに関連する理由により、実行リクエストは特定の HTTP アドレスの PHP インタープリターに送信されます。4D はデフォルトでアドレス 127.0.0.1 そしてポート 8002 を使用します。このアドレスやポートが他のサービスすでに使用されている場合、あるいは同じマシン上で複数のインターパリターが動作する場合、設定を変更する必要があります。これをおこなうには IP アドレスとポート番号 パラメーターを変更します。

HTTP アドレスは 4D と同じマシンでなければならない点に注意してください。

- 外部インターパリター

外部の PHP インタープリターを使用する場合、それは FastCGI でコンパイルされ、4D と同じマシン上になければなりません ([4D で PHP スクリプトを実行する](#) の "異なる PHP インタープリターと php.ini を使用する" 参照)。このオプションを選択すると 4D は内部の PHP インタープリターに接続しなくなります。この設定では、外部インターパリターの制御を開発者がおこなわなくてはならない点に留意してください。

4D Server: この設定は、4D Server と 4D リモートマシン間で共有されます。つまり、サーバーマシンで外部インターパリターを利用する時は、クライアントマシンで内部インターパリターを利用することはできません。もしサーバーがポート 9002 で外部インターパリターを利用する場合には、クライアントマシンでも同じポート番号でインターパリターを利用しなければなりません。

オプション

このエリアのオプションは 4D の PHP インタープリターの自動管理に関するものであり、外部インターパリター オプションが選択されているときは無効になります。

- プロセス数: 4D の PHP インタープリターは "子プロセス" と呼ばれる一連のシステム実行プロセスを起動します。最適化のため、デフォルトで同時に 5 つの子プロセスを保持し実行できます。必要に応じて子プロセスの数を変更できます。たとえば、PHP インタープリターを頻繁に呼び出す場合、数を増やしたいと思うかもしれません。この点に関する詳細は [4D で PHP スクリプトを実行する](#) の "アーキテクチャ" を参照ください。

注: macOS ではすべての子プロセスが同じポートを共有します。Windows では子プロセスごとにポート番号が異なります。最初に使用されるポートは PHP インタープリター用に設定されたポートです。そして次の子プロセスはインクリメントされた番号を使用します。たとえば、デフォルトポート番号が 8002 で 5 つの子プロセスを起動すると、8002 から 8006 が使用されます。

- インターパリターを再起動 X リクエスト後: 4D の PHP インタープリターが受け付けるリクエストの最大数を設定します。この数に達すると、インターパリターが再起動されます。このパラメーターに関する詳細は FastCGI-PHP のドキュメントを参照ください。

注: このダイアログボックスでは、すべての接続されたマシンおよびすべてのセッションのデフォルト値が設定されています。各マシンおよび各セッションで異なる設定を適用するために **SET DATABASE PARAMETER** と **Get database parameter** コマンドを使用できます。**SET DATABASE PARAMETER** コマンドで変更された値はカレントセッションで優先されます。

セキュリティページ

このページでは、データへのアクセスやデータベースの保護に関する設定をおこないます。

注: より一般的な 4D のセキュリティ機能については、[4D Security guide](#) を参照ください。

リモートユーザーアクセス

これらの設定は、シングルユーザーモードで開かれたプロジェクトデータベースには適用されません。

- デザインおよびランタイムエクスプローラーアクセス権: データベースのデザインモードにアクセスし、ランタイムエクスプローラーを表示する権利を特定のグループに付与します。

注:

 - デザインモードへのアクセスグループを設定すると、データ読み込みダイアログのテーブルを作成 オプションが無効となります。このダイアログボックスに関する詳細は [ファイルからデータを読み込む](#) を参照ください。
 - Designer と Administrator は常にデザインモードとランタイムエクスプローラーにアクセスできます。設定されるアクセスグループのメンバーに含まれる必要はありません。ユーザーおよびグループに関する詳細は [ユーザー & グループ](#) を参照ください。
- デフォルトユーザー: デフォルトユーザーが設定されると、データベースを開く、あるいはデータベースにログインするすべてのユーザーは、このデフォルトユーザーに定義されたアクセス権と同じ制限を持つことになります。ユーザー名の入力が不要になるだけでなく、ユーザー名の入力が不要になるだけでなく、デフォルトユーザーにパスワードを割り当てていない場合、パスワードダイアログボックスは表示されず、データベースが直接開かれます。このオプションを使用することで、完全なデータコントロールシステムを維持しつつ、データベースへのアクセスをシンプルにすることができます。
 - デフォルトユーザーにパスワードを割り当てる場合、データベースが開かれるときにダイアログが表示され、パスワードの入力を求められます。
 - デフォルトユーザーにパスワードを割り当たない場合、上記のダイアログは表示されません。

注: "デフォルトユーザー" モードが有効になっているときでも、強制的にユーザー認証ダイアログを表示させることができます。これはたとえば Administrator や Designer としてログインするために必要となります。これには、データベースを開いたり接続したりする際に Shift キーを押したままにします。
- パスワードダイアログにユーザーリストを表示する: このオプションが選択されていると、ユーザー認証ダイアログにユーザーリストが表示され、ユーザーはその中から名前を選択し、パスワードを入力することになります。オプションが選択されていない場合、ユーザーは名前とパスワードの両方を入力します。パスワードダイアログボックスの 2つのバージョンに関する詳細は [アクセスシステムの概要](#) を参照ください。
 - ユーザーリストをABC順で表示する(上記オプションが選択されているときのみ有効です): このオプションが選択されていると、ユーザー認証ダイアログボックスのユーザーリストは名前の ABC 順に表示されます。
- ユーザーは自分のパスワードを変更可能: このオプションが選択されていると、ユーザー認証ダイアログに 変更 ボタンが表示されます。このボタンを使用すると、パスワードを変更するためのダイアログボックスが表示されます(このダイアログに関する詳細は [パスワードアクセスシステムの保守](#) の "ユーザーによるパスワードの変更" を参照ください)。必要であれば 変更 ボタンを非表示にし、パスワードの変更を禁止することができます。それには、このオプションの選択を外します。

オプション

- フォーミュラエディタと4D Write Proドキュメントで使用できるコマンドとプロジェクトメソッドの制限 :

セキュリティのため 4D はデフォルトで、アプリケーションモードの **フォーミュラエディター** においてコマンド、関数、プロジェクトメソッドへのアクセスを制限しています。これは、[ST INSERT EXPRESSION](#) コマンドによって 4D Write Proドキュメントやマルチスタイルエリアに追加されるフォーミュラエディターにおいても同様です。[SET ALLOWED METHODS](#) コマンドを使用して明示的に許可された 4D関数やプロジェクトメソッドのみを使用することができます。以下のオプションを使用して、部分的あるいは全体的にこのフィルタリングを無効にできます。以下のオプションを使用して、部分的あるいは全体的にこのフィルタリングを無効にできます。

 - すべてのユーザーを制限する(デフォルトオプション): Designer と Administrator を含むすべてのユーザーに対し、コマンドや関数、プロジェクトメソッドへのアクセスを制限します。
 - DesignerとAdministratorは制限しない: このオプションは Designer と Administrator のみに、4Dコマンドやメソッドへの完全なアクセスを与えます。他のユーザーには制限をかけつつ、管理者に無制限のアクセスを与えるために使用できます。開発段階では、このモードを使用してすべてのフォーミュラやレポート等を自由にテストできます。運用時には、一時的にコマンドやメソッドへのアクセスを与えるためなどに使用

できます。これをおこなうには、コマンドへのフルアクセスが必要なダイアログを呼び出したり印刷処理を開始したりする前に ([CHANGE CURRENT USER](#) コマンドを使用して) ユーザーを切り替えます。そしてその処理が終了したのちに元のユーザーに戻します。注: 前のオプションを使用してフルアクセスが有効にされると、このオプションは効果を失います。

- 誰も制限しない: このオプションはフォーミュラの制御を無効にします。このオプションが選択されると、ユーザーはすべての 4Dコマンドおよびプログラミングコマンド、さらにはプロジェクトメソッドを使用できます (非表示のものを除く)。注: このオプションは [SET ALLOWED METHODS](#) コマンドより優先されます。このオプションが選択されると、コマンドの効果はなくなります。
- 外部ファイルのユーザー設定を有効にする: 外部ファイル化したユーザー設定を使用するにはこのオプションを選択します。このオプションが選択されると、設定をおこなうダイアログが最大 3つになります: ストラクチャー設定、ユーザー設定、そして データファイル用のユーザー設定 です。詳細は [ユーザー設定](#) を参照ください。
- コンポーネントの "On Host Database Event" メソッドを実行: [On Host Database Event データベースメソッド](#) は 4Dコンポーネントの初期化とバックアップフェーズを容易にします。セキュリティ上の理由から、このメソッドの実行はそれぞれのホストデータベースにおいて明示的に許可されなければなりません。そのためにはこのオプションをチェックします。デフォルトでは、チェックされていません。

このオプションがチェックされていると:

- 4D コンポーネントがロードされます。
- コンポーネントそれぞれの [On Host Database Event データベースメソッド](#) がホストデータベースによって呼び出されます。
- メソッドのコードが実行されます。

このオプションがチェックされていないと:

- 4D コンポーネントはロードはされるものの、初期化とバックアップフェーズはコンポーネントによって管理されなければなりません。
- コンポーネントの開発者は、これらのフェーズ (スタートアップとシャットダウン) 中にホストデータベースによって呼び出されなければならないコンポーネントメソッドを公開する必要があります。
- ホストデータベースの開発者は、コンポーネントの適切なメソッドを適切なタイミングで呼び出さなければなりません (コンポーネントのドキュメンテーションにて解説が必要です)。

互換性ページ

互換性ページには、以前の 4D バージョンとの互換性を管理するためのパラメーターがまとめられています。

表示されるオプションの数は、元のデータベース/プロジェクトが作成されたバージョンや、そのデータベース/プロジェクトでおこなわれた設定の変更により異なります。

このページでは、v18 以降のバージョンから変換された 4D データベース/プロジェクトで利用可能な互換性オプションのみを説明します。それ以前のバージョンから引き継がれる互換性オプションについては [doc.4d.com の互換性ページ](#) を参照ください。

- 旧式ネットワークレイヤーを使用する: 4D v15 のリリース以降、4D アプリケーションは 4D Server とリモートの 4D マシン (クライアント) 間の通信に、*ServerNet* という新しいネットワークレイヤーを使い始めました。以前のネットワークレイヤーは廃止予定となります、既存のデータベースとの互換性を保つために保持されます。このオプションを使用すると、4D Server アプリケーションにおいて、必要に応じていつでも以前のネットワークレイヤーを有効化することができます。*ServerNet* は新規に作成されたデータベースおよび v15 以降から変換されたデータベースにおいては自動的に使用されます (このオプションがチェックされます)。この設定を変更する場合、変更を反映するにはアプリケーションを再起動する必要があります。接続していたクライアントアプリケーションも、新しいネットワークレイヤーで接続するため再起動しなければなりません。注: このオプションは、`SET DATABASE PARAMETER` コマンドを使い、プログラミングによって管理することもできます。
- 標準の XPath を使用: デフォルトでは、v18 R3 より前のバージョンの 4D から変換されたデータベースではチェックが外されており、4D v18 R3 以降で作成されたデータベースではチェックされています。v18 R3 以降、4D の XPath 実装は、より多くの述語に対応しサポートするために変更されました。結果的に、以前の標準でない一部の機能は動作しなくなります。これには以下のよう機能が含まれます:
 - 最初の "/" はルートノードに限らない - "/" を XPath 式の最初の文字として使用しても、ルートノードからの絶対パスの宣言にはなりません。
 - 暗示的なカレントノードはなし - カレントノードは XPath 式の中に含められていなければなりません。
 - 繰り返された構造内の再帰的な検索は不可 - 最初の要素のみが解析されます。

標準的なものでなくとも、コードが以前と同じように動くように以前の機能を保ちたい場合もあるかもしれません。その場合、この チェックを外してください。その一方で、これらの非標準の実装をコード内で使用しておらず、拡張された XPath 機能 ([DOM Find XML element](#) コマンドの説明参照) をデータベース内で利用したい場合、この 標準の XPath を使用 オプションが チェックされている ことを確認してください。

- macOS にて改行コードとして LF を使用する: 4D v19 R2 以降 (XML ファイルについては 4D v19 R3 以降) の新規プロジェクトにおいて、4D は macOS でデフォルトの改行コード (EOL) として CR (xml SAX では CRLF) ではなくラインフィード (LF) をテキストファイルに書き込みます。以前の 4D のバージョンから変換されたデータベースにおいてこの新しい振る舞いを利用したい場合には、このオプションをチェックしてください。詳細については [TEXT TO DOCUMENT](#)、[Document to text](#) および [XML SET OPTIONS](#) を参照ください。
- Unicode テキストファイルに書き込んでいる際にデフォルトで BOM を追加しない: 4D v19 R2 以降 (XML ファイルについては 4D v19 R3 以降)、4D はデフォルトでバイトオーダーマーク (BOM) なしでテキストファイルに書き込みます。以前のバージョンでは、テキストファイルはデフォルトで BOM 付きで書き込まれていました。変換されたプロジェクトでこの新しい振る舞いを有効化するには、このオプションを選択します。詳細については [TEXT TO DOCUMENT](#)、[Document to text](#) および [XML SET OPTIONS](#) を参照ください。
- フィールド作成時にデフォルトで "ヌル値を空値にマップ" オプションのチェックを外す: ORDA の仕様により合致するために、4D v19 R4 以降で作成されたデータベースにおいては、フィールド作成時に ヌル値を空値にマップ フィールドプロパティがデフォルトでチェックされなくなります。このオプションにチェックを入れることで、変換されたデータベースにおいてもこのデフォルトの振る舞いを適用することができます (ORDA で NULL 値がサポートされるようになったため、今後は空値ではなく NULL 値の使用が推奨されます)。

一般ページ

このページには、4Dアプリケーションの一般的な動作を設定するためのオプションが含まれています。

オプション

開始時

このオプションは、ユーザーがアプリケーションのみを起動したとき、4D が起動時に提供するデフォルトの表示を設定することができます。

- 何もしない: アプリケーションウィンドウのみが表示されます。
- ローカルプロジェクトを開くダイアログ: 4Dは標準のドキュメントを開くダイアログボックスを表示し、ローカルのプロジェクトを選択することができます。
- 最後に使用したプロジェクトを開く: 4D は最後に使用されたプロジェクトを直接開きます。ドキュメントを開くダイアログボックスは表示されません。
このオプションが選択されているときに、ドキュメントを開くダイアログボックスを強制的に表示させるには、プロジェクトを起動する際に、Alt (Windows) または Option (macOS) キーを押します。
- リモートプロジェクトを開くダイアログ: 4D は 4D Server にログオンする標準のダイアログボックスを表示し、ネットワークに公開されたプロジェクトを指定することができます。
- Welcomeウィザードを開くダイアログ (初期設定): 4D は Welcomeウィザードダイアログボックスを表示します。

4D Server: 4D Server アプリケーションは、このオプションを無視します。この環境においては、何もしないモードが常に選択されます。

自動フォーム作成

このオプションは、バイナリデータベースでのみ使用され、プロジェクトアーキテクチャーでは無視されます。doc.4d.com を参照ください。

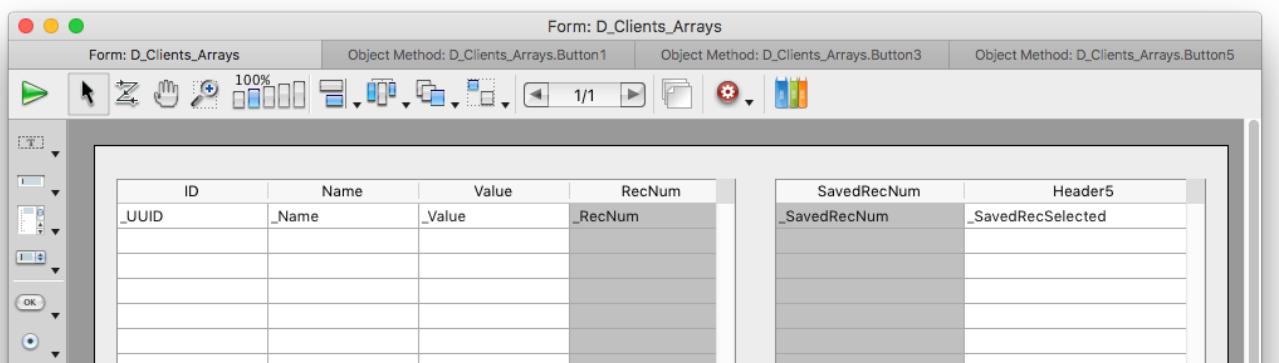
ウィンドウのタブ (macOSのみ)

macOS Sierra 以降、Mac のアプリケーションは、複数のウィンドウを整理しやすくする自動ウィンドウタブ機能を利用することができます。単一の親ウィンドウ内でドキュメントウィンドウを積み重ね、タブを通してブラウズすることができます。この機能は小さなスクリーンや、トラックパッドを使用している場合などに有用です。

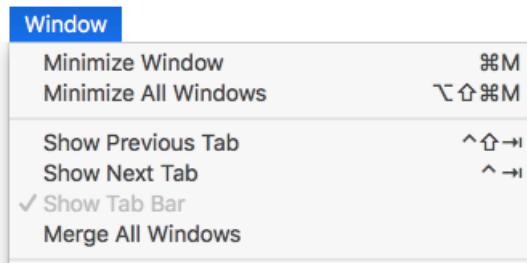
この機能は、以下の環境において利用することができます (4D 64-bit版のみ):

- メソッドエディターウィンドウ
- フォームエディターウィンドウ

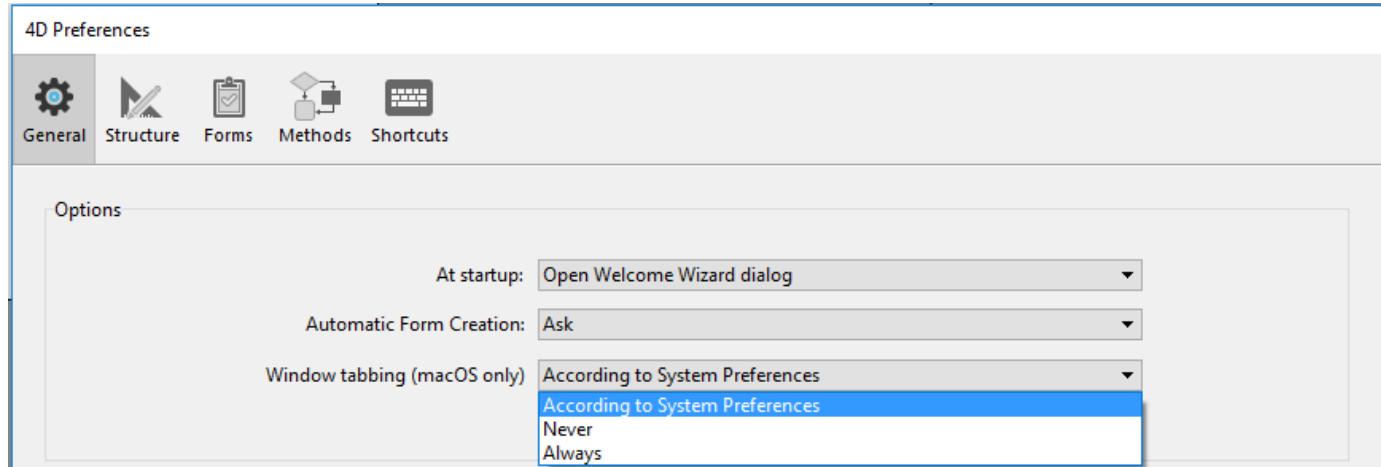
これらのエディターのウィンドウはすべて、タブ形式にすることができます:



ウィンドウ メニューのコマンドによって、タブを管理することができます:



4D の環境設定ダイアログボックス内では、ウィンドウタブ オプションでこの機能を管理することができます:



次の値が提供されています:

- システム設定に従う (デフォルト): 4D のウィンドウは、macOSシステム環境設定で定義されているように振る舞います (フルスクリーン時のみ、常に、あるいは手動)。
- しない: 4Dフォームエディターあるいはメソッドエディターで開かれた新しいドキュメントは常に新しいウィンドウを作成します (タブは作成されません)。
- 常に: 4Dフォームエディターあるいはメソッドエディターで開かれた新しいドキュメントは常に新しいタブを作成します。

アピアランス (macOSのみ)

このメニューで、4D開発環境で使用するカラースキームを選択します。指定されたカラースキームは、デザインモードのすべてのエディターとウィンドウに適用されます。

デスクトップアプリケーションで使用するカラースキームは、ストラクチャー設定ダイアログボックスの "インターフェース" ページで設定することができます。

次の値が提供されています:

- システムのカラースキーム設定に合わせる (デフォルト): macOSシステム環境設定で定義されているカラースキームを使用します。
- Light: ライトテーマを使用します。
- Dark: ダークテーマを使用します。

この設定は macOS でのみサポートされています。Windows上では、"Light" テーマが常に使用されます。

アプリケーションモードに移動する時に、デザインモードを終了する

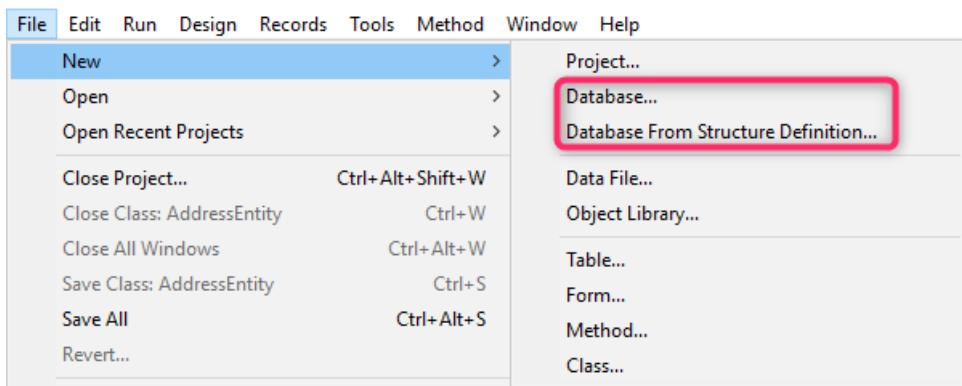
このオプションが選択されていると、アプリケーションモード コマンドを使用してユーザーがアプリケーションモードに移動する際に、デザインモードのすべてのウィンドウが閉じられます。このオプションがチェックされていないと (初期設定)、デザインモードのウィンドウはアプリケーションモードの後ろに表示されたままになります。

バイナリー形式のデータベース作成を有効化する

このオプションをチェックすると、ファイル > 新規 メニューと、ツールバーの 新規 ボタンに、2つの項目が追加されます:

- データベース...

- ストラクチャー定義を使用したデータベース...



この項目を使用するとバイナリーデータベースを作成することができるようになります([新しいデータベースを作成する](#) の章を参照)。今後 4D は、新規の開発にはプロジェクトベースのアーキテクチャーを使用することを推奨するからです。

新規プロジェクト作成時

ログファイルを使用

このオプションをチェックすると、新規データベース作成時にログファイルが自動的に開始され、使用されます。詳細な情報については [ログファイル \(.journal\)](#) を参照ください。

パッケージを作成する

このオプションがチェックされていると、4Dデータベースは自動で .4dbase 拡張子が付いたフォルダーに作成されます。

この原則のため、macOS ではデータベースフォルダーが専用プロパティ付きのパッケージとして表示されます。Windows では、これは普通のフォルダーと変わりありません。

Project ソースファイルにトークンを含める

このオプションを有効にすると、新規の 4Dプロジェクトで保存された [メソッドのソースファイル](#) には、クラシックランゲージおよびデータベースオブジェクト（定数、コマンド、テーブル、フィールド）用の トークン が含まれます。トークンとは、ソースコードファイルに挿入される :C10 や :5 などの追加文字で、テーブルやフィールドの名前を変更したり、4Dバージョンに関係なく要素を識別したりすることを可能にします（[フォーミュラ内のトークンの使用](#) を参照ください）。

バージョン管理システムや外部のコードエディターを新規プロジェクトで使用したい場合、これらのツールでのコードの可読性のために、このオプションのチェックを外すことができます。

このオプションは、プロジェクトにのみ適用できます（バイナリーデータベースでは常にトークンが含まれます）。

`option` パラメーターに 1 を指定して `METHOD GET CODE` を呼び出すと、トークンを含むコードをいつでも取得することができます。

既存プロジェクトからトークンを除外する

テキストエディターを使い、`<applicationName>.4DProject` ファイルに以下のキーを挿入することで、既存のプロジェクトでも トークンなし でコードを保存することができます：

```
"tokenizedText": false
```

この設定は、メソッドが保存されるときにのみ考慮されます。つまり、再保存しない限り、プロジェクト内の既存メソッドはそのまま残されます。

`.gitignore` ファイルを作成する

新しいプロジェクトでは、いくつかのファイルを git に無視させたいことがあるかもしれません。

この設定をおこなうには、`.gitignore` ファイルを作成する オプションをチェックします。

このボックスがチェックされている場合、4D でプロジェクトを作成すると、4D は `.gitignore` ファイルを `Project` フォルダーと同階層に作成します ([プロジェクトのアーキテクチャー 参照](#))。

鉛筆アイコンをクリックすると、`.gitignore` ファイルのデフォルトの内容を定義することができます。これにより、`.gitignore` 設定ファイルがテキストエディターで開かれます。このファイルの内容は、新規プロジェクトで `.gitignore` ファイルを生成する際に使用されます。

`.gitignore` ファイルの仕組みを理解するには、[git の公式ドキュメント](#) が参考になります。

テキスト比較の言語

このパラメーターは新規データベースにおいて、文字列の処理と比較で使用されるデフォルトの言語を設定します。言語の選択は、テキストの並べ替えや検索、文字の大小などの比較ルール等に直接影響を与えます。ただし、テキストの翻訳や日付・時刻・通貨のフォーマットはシステムの言語設定が使用され、この設定には影響されません。初期設定では、4D はシステムに設定されているカレントのユーザー言語を使用します。

つまり、4Dデータベースはシステム言語とは異なる言語で動作することができます。データベースが開かれるとき、4Dエンジンはデータファイルに使用されている言語を検知し、(インタープリターやコンパイルモードの) ランゲージに提供します。データベースエンジン、あるいはランゲージのいずれがテキスト比較をおこなうかに関わらず、同じ言語が使用されます。

新規にデータファイルを作成する際、4D はこのメニューで設定されている言語を使用します。ストラクチャーの言語と異なる言語のデータファイルを開くと、データファイルの言語が使用され、ストラクチャーに言語コードがコピーされます。

データベース設定を使用して、開かれているデータベースの言語を変更することができます ([テキスト比較 参照](#))。

ドキュメントの場所

このエリアでは、カレントブラウザーに表示される 4D HTMLドキュメントへのアクセスを設定します:

- メソッドエディターで、4Dクラス関数またはコマンド名にカーソルがあるときに、F1キーを押したとき
- エクスプローラーの コマンドページ 上の 4Dコマンドをダブルクリックしたとき

ドキュメント言語

表示する HTMLドキュメントの言語。アプリケーションの言語とは別のドキュメント言語を選択することができます。

最初にローカルフォルダーを見る

このオプションは、コマンドドキュメントへのアクセスに関してのみ考慮されます (クラス関数を除く)。

4Dがドキュメントのページを探す場所を設定します。

- チェックされている場合 (デフォルト)、4D はまずローカルフォルダーでページを探します (後述参照)。ページが見つかれば、4D はそのページをカレントブラウザーで表示します。ページが見つかれば、4D はそのページをカレントブラウザーで表示します。この場合インターネットに接続されていない環境でも、ローカルのドキュメントが参照できます。
- チェックされていない場合、4D はオンラインドキュメントの Webサイトに直接アクセスし、カレントブラウザーでページを表示します。ページが見つからない場合、4D はブラウザーにエラーメッセージを表示します。

ローカルフォルダー

このオプションは、コマンドドキュメントへのアクセスに関してのみ考慮されます (クラス関数を除く)。

スタティックな HTMLドキュメントの場所を指定します。デフォルトでこれは `¥Help¥Command¥language` サブフォルダーに設定されています。このエリアに割り当てられているメニューをクリックすると、場所を見ることができます。このサブフォルダーが存在しない場合、場所は赤で表示されます。

この場所は必要に応じて変更することができます。たとえば、アプリケーションの言語とは異なる言語でドキュメントを表示したい場合などです。HTMLドキュメントは、異なるボリュームや Webサーバー上などに置くことも可能です。他の場所を指定するには、メニューの隣の [...] ボタンをクリックし、ドキュ

ントのルートフォルダー (`fr` , `en` , `es` , `de` または `ja` などの言語に対応するフォルダー) を選択します。

ストラクチャーページ

プライマリーキー

環境設定内のこれらのオプションによって、新しくテーブルが追加されたとき、または [プライマリーキー管理](#) 機能の使用によって 4D が自動的に追加するプライマリーキーのデフォルトの名前と型を変更することができます。

次のオプションから選択することができます:

- Name (デフォルトでは "ID"): プライマリーキーのフィールドのデフォルト名を設定します。 [4D の命名規則](#) に従う範囲内であればどんな名前も使用できます。
- デフォルトタイプ (デフォルトでは [倍長整数](#)): プライマリーキーフィールドのデフォルトの型を設定します。 UUID を選択することもできます。この場合、デフォルトで作成されたプライマリーキーフィールドは [文字型](#) となり、UUIDフォーマットと自動UUIDプロパティにチェックが入っています。

ストラクチャーエディター

このオプショングループでは、4Dストラクチャーエディターの表示を設定します。

ストラクチャの描画クオリティ

このオプションで、ストラクチャーエディターの描画レベルを変更できます。デフォルトで品質は [高](#) に設定されています。標準品質を選択して、表示速度を優先させることができます。この設定の効果は主にズーム機能を使用する際に実感することができます ([ストラクチャーエディター](#) のズーム参照)。

フォルダーが表示対象外のとき

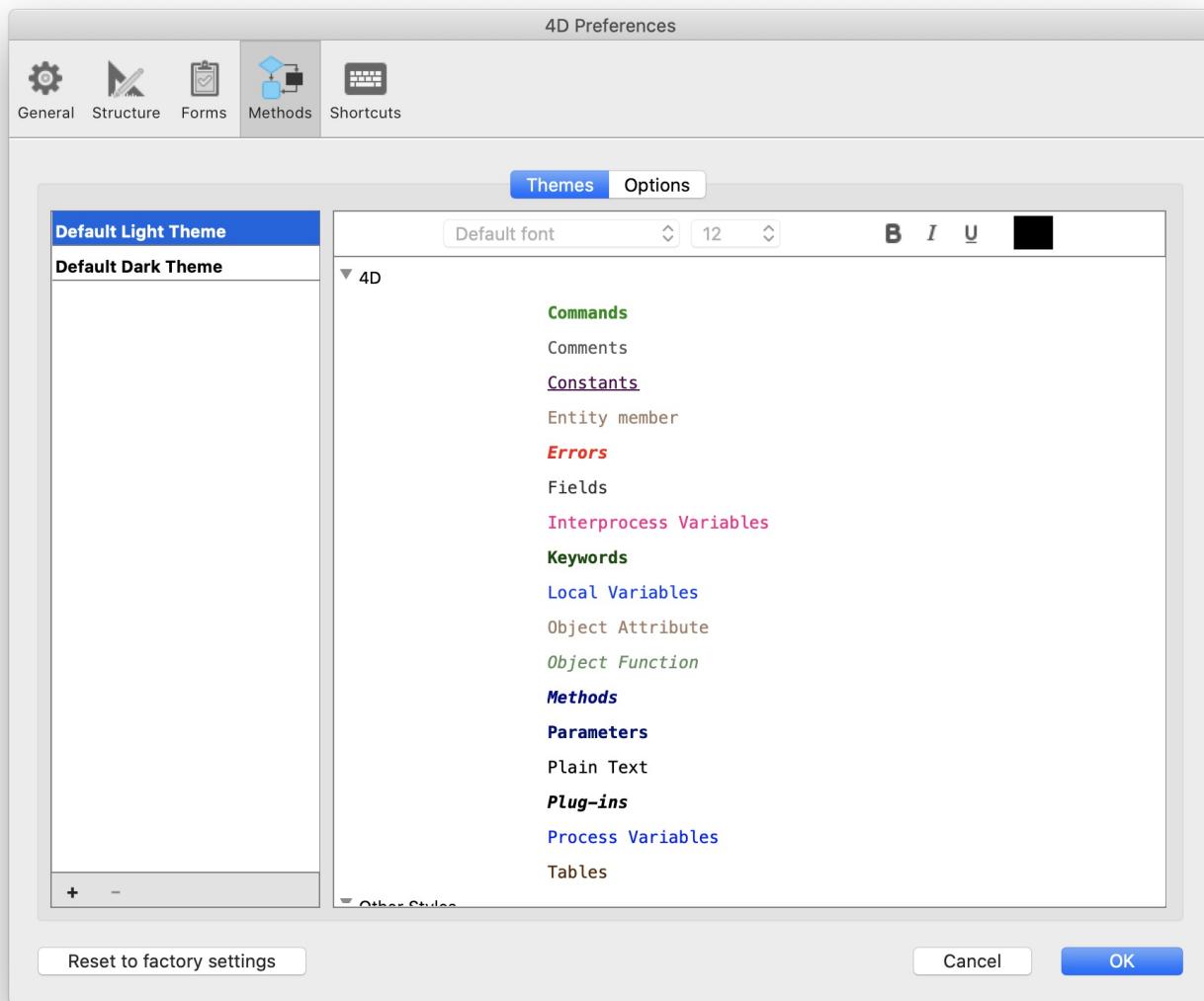
このオプションは、ストラクチャーエディターでフォルダーによって選択する際の、テーブルが表示対象外のときのアピアランスを設定できます ([フォルダーごとにテーブルをハイライト/薄暗くする](#) 参照)。薄暗く表示 (テーブルイメージの代わりに影が表示される) または非表示 (テーブルは完全に見えなくなる) が選択できます。

メソッドページ

このページでは、メソッドエディターのインターフェースやデフォルトの表示、および動作に関するオプションを設定します。ページはテーマとオプションという2つのタブに分けられています。

テーマ

このページでは、メソッドエディターのテーマを選択・作成・設定することができます。テーマは、コードエディターに表示される項目のフォント、フォントサイズ、カラー、スタイルを定義します。



テーマリスト

このリストでは、コードエディターに適用するテーマを選択します。利用可能なテーマがすべて表示され、カスタムテーマがある場合はそれも表示されます。4Dはデフォルトで2つのテーマを用意しています：

- デフォルトのLightテーマ
- デフォルトのDarkテーマ

デフォルトのテーマは変更や削除ができません。

以前の4Dリリースで、メソッドエディターのスタイルをカスタマイズしていた場合、myThemeテーマが自動的に追加されます。

カスタムテーマの作成

完全にカスタマイズ可能なテーマを作成することができます。テーマを作成するには、既存のテーマを選択して、テーマリストの下部にある+ をクリックします。また、4D Editor Themes フォルダー内のテーマファイルをコピーして、カスタマイズしたテーマを追加することもできます（後述参照）。

カスタムテーマファイル

カスタムテーマは、それぞれ別の *themeName.json* という JSONファイルに格納されます。カスタムテーマの JSONファイルは、4D preferences ファイルと同じ階層にある 4D Editor Themes フォルダーに格納されます。

カスタムテーマでキー値が定義されていない場合は、デフォルトのLightテーマ の値がデフォルトとなります。JSONテーマファイルが無効な場合、デフォルトのLightテーマ が読み込まれ、エラーが発生します。

外部エディターでテーマファイルを変更した場合は、変更内容を反映させるために 4Dを再起動する必要があります。

テーマの定義

テーマを定義するとは、以下のことを意味します：

- コードエディター全体のグローバルフォントとフォントサイズを設定する。
- 4D のランゲージ要素 (フィールド、テーブル、変数、引数、SQL など)、SQL のランゲージ要素 (キーワード、関数など)、そして背景色のそれぞれにスタイルと色を割り当てる。

異なる色やスタイルを組み合わせることは、コードのメンテナンス目的に特に便利です。

フォントとフォントサイズ

フォントとフォントサイズ のメニューで、すべてのカタゴリーのメソッドエディターの入力エリアで使用するフォント名とサイズを選択できます。

4D ランゲージと SQL ランゲージ

ランゲージ要素の種類ごとに、異なるフォントスタイルやカラー (フォントカラーと背景色) を設定できます。カスタマイズする要素は、カタゴリーリストで選択できます。

その他のスタイル

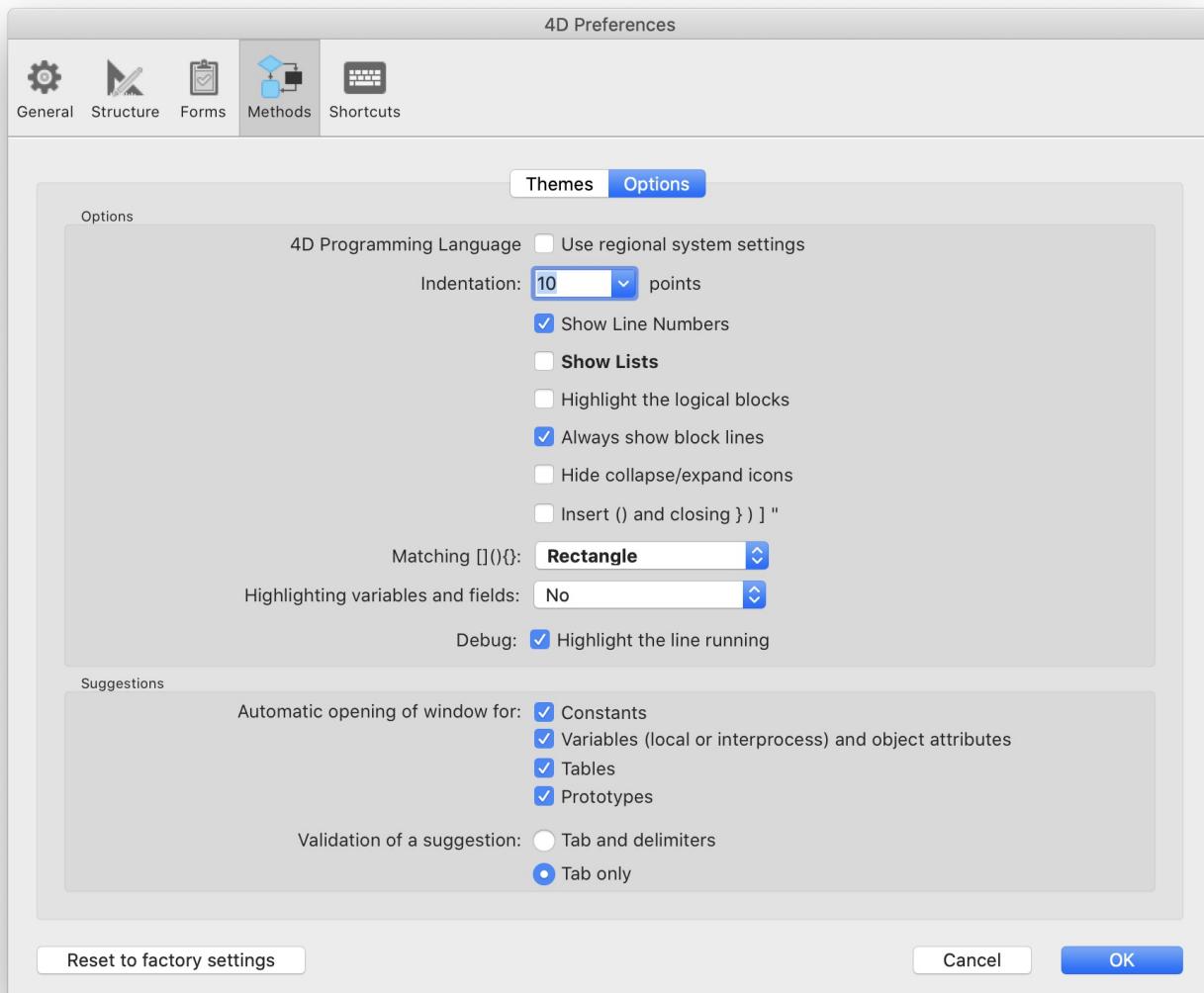
これらのオプションは、メソッドエディターとデバッガーアンターフェースで使用される様々なカラーを設定します。

Category	Element
> 4D Language	
▼ Other Styles	
	Background color
	Cursor line background color
	Highlight of the running line in the debugger
	Execution line background color
	Border of the running line in the debugger
	Highlight background color
	Highlight of the blocks
	Highlight of the parentheses
	Highlight of the found words
	Suggested text
	Selection back color
> SQL Language	

	説明
背景色	メソッドエディターウィンドウの背景色。
デバッガ内の実行行の境界線	オプション ページで "実行行をハイライト" オプションが有効になっている場合、デバッガーで現在実行中の行を囲む境界線の色。
カーソル行背景色	カーソルのある行の背景色。
実行行背景色	デバッガーで実行中の行の背景色。
検索で見つかった単語のハイライト	検索して見つかった単語のハイライト色。
カッコのハイライト	対応するカッコのハイライト色 (カッコのペアがハイライト表示される時に使用されます。 オプション 参照)。
ブロックのハイライト	オプション で "論理ブロックを強調" オプションが有効化されていた場合の、選択された論理ブロックのハイライト色。
同じ変数やフィールドのハイライト	オプション で "変数とフィールドを強調" オプションが有効になっている場合、同じ変数またはフィールドテキストの他の出現箇所のハイライトカラー。
デバッガ内の実行行のハイライト	オプション で "実行行をハイライト" が有効になっている場合、デバッガーで現在実行中の行のハイライトカラー。
選択範囲の背景色	選択範囲の背景色。
サジェストテキスト	メソッドエディターで表示されるオートコンプリートテキストの色。

オプション

このページでは、メソッドエディターの表示オプションを設定します。



オプション

4Dプログラミングランゲージ (リージョンシステム設定を使う)

ローカル4Dアプリケーション用の "国際的な" コード設定を有効化/無効化することができます。

- チェック無し (デフォルト): 4Dメソッドにおいて English-US設定と英語でのプログラミングランゲージが使用されます。
- チェック有り: リージョン設定が使用されます。

このオプションを変更した場合、変更を反映するには 4Dアプリケーションを再起動する必要があります。

インデント

メソッドエディターで、4Dコードが使用するインデントの値を設定します。値はポイントで設定します (デフォルトは 10)。

4D は構造を明確にするために自動でコードをインデントします:

```

1  □ If ($vListItemPos#0)
2    // Get the list item information
3    GET LIST ITEM(hList;$vListItemPos;$v
4      // Is the item a Department item?
5    □ If ($vListItemRef ?? 31)
6      // If so, it is a double-click
7      ALERT("You double-clicked on the
8    □ Else
9      // If not, it is a double-click
10     // Using the parent item ID to
11     $vDepartmentID:=List item parent
12     QUERY([Departments];[Departments]
13       // Tell where the Employee is
14       ALERT("You double-clicked on the
15     End if
16   End if
17
18  standard indentation

```

アルゴリズムが複雑になり、階層レベル数が増えた場合に、この値を変更すると便利な場合があります。具体的には、この値を減らすことで水平スクロールを減らすことができます。

行番号を表示

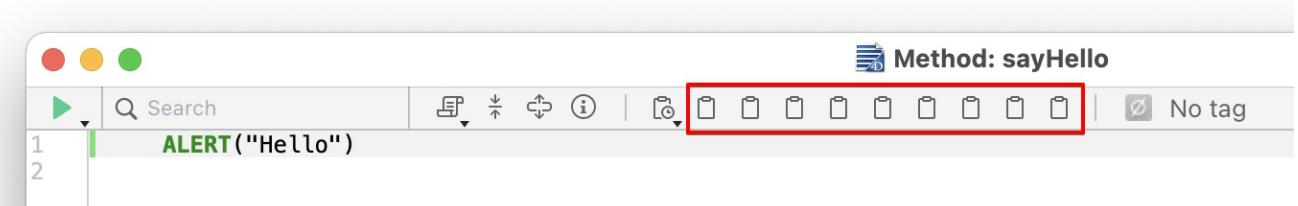
メソッドエディターで、行番号をデフォルト表示するかどうかを設定します。メソッドエディターにて、直接の行番号を表示/非表示設定することもできます。

リスト表示

メソッドエディターウィンドウを開いたときに、オブジェクト（コマンド、テーブル、フィールド等）のリストをデフォルトで表示するかどうかを設定します。メソッドエディターで直接このリストを表示/非表示にすることもできます。

クリップボード表示

コードエディターに複数のクリップボードを表示するかどうかを選択できます。



これらのクリップボードが非表示の場合でも、対応する [クリップボードショーカット](#) は有効です。

論理ブロックを強調

チェックされている場合、展開されたノードの上にマウスを置いたときに論理ブロック（たとえば If/End if など）に含まれているコードがすべてハイライトされます：

```

12  □ If (<>PS_EditMovies=0)
13    | <>PS_EditMovies:=New process ($CurrentMethName;
14    □ Else
15      BRING TO FRONT(<>PS_EditMovies)
16    End if
17

```

ハイライトカラーは [テーマ](#) ページにて設定が可能です。

ブロック行を常に表示

垂直のブロック線を常に非表示にできます。ブロック線はノードを視覚的に繋ぐためのものです。デフォルトでは、これらは常に表示されています（ただし展開/折りたたみアイコンが非表示の場合は除きます。以下参照）。

9	└ If (Count	9	└ If (Count
10		10	
11	\$CurrentN	11	\$CurrentN
12	└ If (<>PS_	12	└ If (<>PS_
13	<>PS_Edi:	13	└ <>PS_Edi:
14	└ Else	14	└ Else
15	BRING_TC	15	BRING_TC
16	End if	16	End if
17		17	
18	└ Else	18	└ Else
19		19	

折りたたみ/展開アイコンを隠す

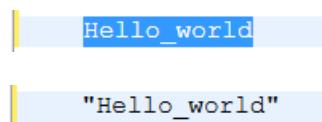
コードを表示する際に、展開/折りたたみアイコンをデフォルトで非表示にすることができます。このオプションがチェックされているとき、ノードアイコン（とブロック線。前述参照）は、マウスがノードの上に置かれた際にのみ一時的に表示されます：

9	If (Count	9	└ If (Count
10		10	
11	\$CurrentN	11	\$CurrentN
12	└ If (<>PS_	12	└ If (<>PS_
13	<>PS_Edi:	13	└ <>PS_Edi:
14	└ Else	14	└ Else
15	BRING_TC	15	BRING_TC
16	End if	16	End if
17		17	
18	└ Else	18	└ Else
19		19	

() と対応する閉じる })] " を自動で挿入

コード入力中に () と、閉じるカッコを自動的に挿入するようにします。このオプションでは 2つの自動機能を管理します：

- ()カッコのペア：4Dコマンド、キーワード、プロジェクトメソッドが提案リストあるいは補完リストから挿入される時、その挿入要素が一つ以上の引数を必須としている場合に追加されます。たとえば、"C_OB" と入力して Tabキーを押すと、4D は自動的に "C_OBJECT()" と表示し、カーソルを () の内部に設定します。
- 閉じる },),], ": {, (, [,あるいは " などの開くカッコを入力した時に、対応する閉じるカッコが追加されます。この機能により、カーソル位置に、あるいは選択されたテキストを囲むように、対応するカッコ記号を挿入できるようになります。たとえば、文字列をハイライトして单一の " を入力すると、選択された文字列全体が "" で囲まれます：



Matching [](){}]

コード中での対応する括弧を強調する方法を設定します。この強調は、括弧（大カッコ[]、中カッコ{}、小カッコ()）が選択されたときに表示されます。次のオプションから選択することができます：

- なし：強調なし
- 四角（デフォルト）：括弧が黒い四角で囲まれます。
`INSERT MENU ITEM [main bar;-1;Get indexed string (79;1);FileMenu]`
- 背景色：括弧がハイライトされます（色は テーマ ページで設定します）。
- 太字：括弧が太字で表示されます。

変数とフィールドを強調

開かれたメソッドウィンドウ内で、同じ変数やフィールド等のオカレンスをすべてハイライトします。

```

4      C_LONGINT (<>PS_EditMovies)
5      C_LONGINT ($Window)
6      C_TEXT ($CurrentMethodName)
7      C_LONGINT ($Win)
8
9      If (Count parameters=0)
10
11     $CurrentMethodName:=Current method
12     If (<>PS_EditMovies=0)
13         <>PS_EditMovies:=New process ($C
14     Else
15         BRING TO FRONT (<>PS_EditMovies)
16     End if

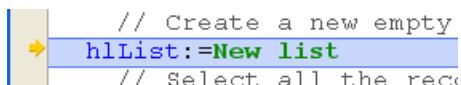
```

- しない(デフォルト): ハイライトなし
- カーソル上のみ: テキストがクリックされた際にのみハイライトされます。
- 選択範囲上のみ: テキストが選択された際にのみハイライトされます。

ハイライトカラーは [テーマ](#) ページにて設定が可能です。

デバッグ (実行行をハイライト)

通常の黄色の矢印インジケーターに加え、デバッガーで実行中の行をハイライトするかどうかを設定します。



```

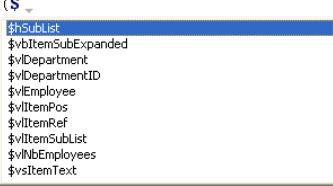
// Create a new empty
h1List:=New list
// Select all the rec

```

このオプションの選択を解除すると、黄色の矢印のみが表示されます。

提案

このエリアでは、メソッドエディターの自動補完メカニズムを設定して、作業習慣に合わせることができます。

	説明
ウィンドウを自動で開く	<p>次の要素に関する提案ウィンドウを自動で開くかを指定します:</p> <ul style="list-style-type: none"> 定数
<ul style="list-style-type: none"> 変数(ローカルまたはインタープロセス)あるいはオブジェクト属性 テーブル プロトタイプ (例: クラス関数) <p>たとえば、"変数(ローカルまたはインターパロセス)あるいはオブジェクト属性" オプションがチェックされている場合、\$ 文字を入力すると提案されるローカル変数のリストが表示されます:</p>  <pre>GET LIST ITEM(\$ \$vSubList \$vItemSubExpanded \$vDepartment \$vDepartmentID \$vEmployee \$vItemPos \$vItemRef \$vItemSubList \$vNbEmployees \$vItemText</pre> <p>対応するオプションのチェックを外すことで、要素ごとにこの機能を無効にできます。</p>	
提案の決定	<p>メソッドエディターで、自動補完ウィンドウに表示されたカレントの提案を受け入れるための、入力コンテキストを設定します。</p> <ul style="list-style-type: none"> タブと区切り文字 このオプションが選択されると、タブキーまたは現在のコンテキストに関連する区切り文字で、現在選択されている提案を決定することができます。たとえば "ALE" と入力して "(" を入力すると、4Dは自動で "ALERT(" とエディターに書き込みます。区切り文字は以下の通りです: (; : = < [{ タブのみ このオプションが選択されると、現在の提案はタブキーを押したときにのみ受け入れられます。これは特に \${1} のように、要素名に区切り文字を入力することを容易にします。 <p>注記: ウィンドウ内をダブルクリックするか、改行キーを押すことで提案を受け入れることもできます。</p>

ショートカットページ

このページには 4D のデザインモードで使用されるすべてのショートカットがリストされています (コピー (Ctrl+C/Command+C) などのシステムショートカットを除く)。

4D Preferences

General Structure Forms Methods Shortcuts

Shortcuts

Actions	Cmd	Shift	Alt	Ctrl	Letter
Query > Query	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Y
Order By	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Y
Close Project	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	W
Close Window	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	W
Close All Windows	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	W
Start Of Block	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	up arrow
Move Lines Up	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	up arrow
Show Current Table	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	U
List of Tables	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	U
4D Server Administration	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	U
Show Next Tab	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Tab
Show Previous Tab	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Tab
Tool Box > Tool Box	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	T
Swap Expression	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	T
Save Window	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	S
Save All	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	S
Flush Data Buffers	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	S
Method	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	R
Apply Formula	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	R
Quit	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Q
Quit and Keep Windows	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Q
Page Setup	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	P
Print	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	P

Reset to factory settings Cancel OK

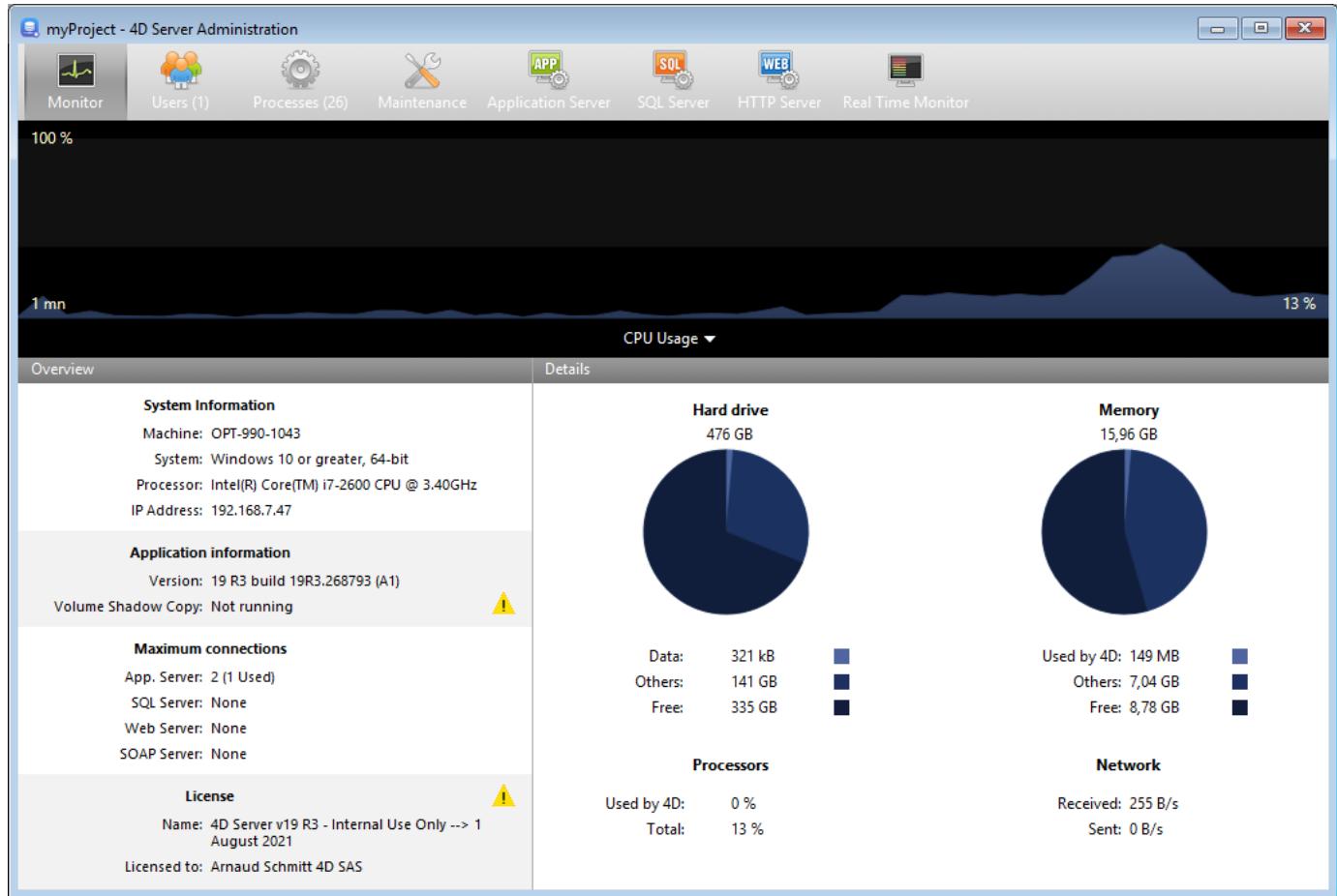
ショートカットを変更するには、設定する項目のモディファイアキー (Shift および Alt) および文字をリスト中で変更します。またダブルクリックすると専用のダイアログボックスでショートカットの編集をおこなうことができます:

それぞれのショートカットには暗黙に Ctrl (Windows) または Command (macOS) キーが含まれることに留意してください。

このリストを編集すると、カスタムのショートカット設定は [ユーザー Preferences ファイル](#) と同じ階層にある *4D Shortcuts vXX.xml* ファイルに保存されます。そのため、4Dがアップデートされても、ショートカット環境設定は残されます。

モニターページ

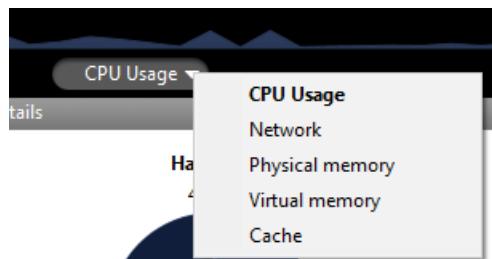
モニター ページには、データベース利用に関する動的な情報のほか、システムや 4D Server アプリケーションの情報が表示されます。



Windows では、このページに表示されるシステム情報の一部は、Windows パフォーマンスアナライザー (WPA) ツールを介して取得されます。これらのツールは、4D Server を起動したセッションを開いたユーザーが、必要な管理権限を持っている場合にのみアクセスできます。

グラフィックエリア

グラフィックエリアでは、複数のパラメーター (CPU 使用率、ネットワークトラフィック、およびメモリ) の変化がリアルタイムで表示されます。ウィンドウの中央にあるメニューから表示する内容を選択します:



- CPU 使用率: すべてのアプリケーションによるマシンの全体的な CPU 使用率。この使用率のうちの 4D Server による使用分は、"プロセッサー" 情報エリアで提供されます。
- ネットワーク: マシン (サーバーまたはクライアント) が 1 秒あたりに受信したバイト数。送信バイト数は "ネットワーク" 情報エリアで提供されます。
- 物理メモリ: 4D Server が使用する、マシンの RAM の量。メモリの利用に関するより詳細な情報は "メモリ" 情報エリアで提供されます。
- 仮想メモリ: 4D Server アプリケーションが使用する仮想メモリの量。このメモリは、アプリケーションのニーズに応じてシステムにより割り当てられます。エリアの右下に表示される値は、現在使用されているメモリ量を示します。左上に表示される値は、利用可能な仮想メモリの最大値を示します。最大値は、アプリケーションの一般メモリ設定に基づき動的に計算されます。
- キャッシュ: 4D Server アプリケーションが使用するキャッシュメモリの量。エリアの右下に表示される値は、現在使用されているメモリ量を示します。

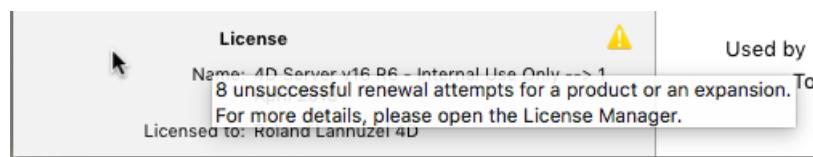
左上に表示される値は、ストラクチャー設定にて指定されたキャッシュメモリの合計サイズを示します。

このオプションが選択されている場合、キャッシュの有効な解析をおこなうために長めの観測時間が必要となるため、グラフエリアのスクロールは遅くなります。

概要エリア

"概要" エリアでは、システム、アプリケーション、そして 4D Server マシンにインストールされたライセンスに関する様々な情報が提供されます。

- システム情報: サーバーのマシン、システムおよび IPアドレス
- アプリケーション情報: 4D Server の内部バージョン番号およびボリュームシャドウコピーステータス
- 最大接続数: サーバータイプ毎に可能な同時接続数
- ライセンス: ライセンスの詳細。プロダクトライセンス、あるいは付随エクスパンションのいずれかが 10日以内に失効するとき（例: サブスクリプション型ライセンスなど）、4D Server は自動的にそのライセンスを 4Dユーザー帳から更新しようとします。この場合、なんらかの理由（接続エラー、無効なアカウント状態、契約が延長されていないなど）で自動更新が失敗した場合、サーバー管理者に警告を伝えるアイコンがライセンスの隣に表示されます。エリア上にマウスをホバーさせると、ライセンス更新状態についての追加の情報が tips として表示されます:



こういった場合には通常、[ライセンスマネージャー](#) をチェックする必要があります。

詳細エリア

"詳細" エリアは、すでにグラフィックエリアで表示されている情報の一部と、追加の情報を提供します。

- ハードディスク: ハードディスク全体、およびデータベースデータ（データファイルとインデックスファイル）の使用スペース、他のファイルの使用スペース、空きスペースなどを表示します。
- メモリ: マシンにインストールされた RAMメモリ、4D Server による使用量、他のアプリケーションによる使用量、および空き容量。4D Server が使用するメモリはグラフィックエリアにも動的に表示できます。
- プロセッサー: 4D Server と他のアプリケーションによる、プロセッサーの使用率。この使用率は絶えず再計算されます。4D Server による使用率はグラフィックエリアにも動的に表示できます。
- ネットワーク: マシン（サーバーまたはクライアント）が受信および送信したその瞬間のバイト数。この値は絶えず更新されます。受信したバイト数はグラフィックエリアにも動的に表示できます。

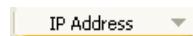
ユーザーページ

ユーザー ページには、サーバーに接続しているユーザーが表示されます:

Employees - 4D Server Administration							
4D User	Machine name	Session name	IP Address	Login date	CPU Time	Activity	
Designer	WIN7-ESMITH	esmith	localhost	03/05/2016 17:24	00:00:02	2%	
Designer	MACWIN7	Arnaud	192.168.10.11	03/05/2016 17:27	00:00:01	0%	

"ユーザー" ボタンには、データベースに接続中のユーザ数が括弧内に表示されます (この番号は、ウィンドウに適用される表示フィルターを考慮しません)。このページには、動的な検索エリアやコントロールボタンもあります。ヘッダーエリアをドラッグ & ドロップして、列の順番を入れ替えることができます。

また、ヘッダーをクリックすると、リストの値が並べ替えられます。クリックするごとに昇順/降順が入れ替わります。



ユーザーリスト

サーバーに接続したユーザーごとに、以下の情報がリストに表示されます:

- システム: クライアントマシンのシステム (macOS/Windows)。
- 4Dユーザー: 4Dユーザー名、またはユーザー マシン上で `SET USER ALIAS` コマンドで設定されていればエイリアス。パスワードシステムが有効になっていない場合、かつエイリアスも設定されていなければ、すべてのユーザーは "Designer" となります。
- マシン名: リモートマシンの名前。
- セッション名: リモートマシン上で開かれたセッション名。
- IP アドレス: リモートマシンの IP アドレス。
- ログイン日: リモートマシンが接続した日付と時刻。
- CPU時間: 接続してからこのユーザーが消費した CPU の時間
- Activity: 4D Server がこのユーザーのために使用する時間の割合 (動的表示)。リモートマシンがスリープモードに切り替わっている場合には "スリープ中" と表示 (以下参照)。

スリープ中ユーザーの管理

4D Server は、サーバーマシンへのアクセスがアクティブである間にスリープモードへと切り替わってしまった 4Dリモートアプリケーションを実行しているマシンについて、特別な管理をします。この場合、接続されている 4Dリモートアプリケーションはこの急な切断を 4D Server へと自動的に知らせます。サーバー側では、接続しているユーザーのアクティビティステータスを スリープ中 へと変更されます:

testServer - 4D Server Administration							
4D User	Machine name	Session name	IP Address	Login date	CPU Time	Activity	
Designer	macmini-program	program	192.168.18.11	20/10/15 09:43	00:00:18	Sleeping	
Designer	iMac-VTalbot-0833	Vanessa Talbot	localhost	20/10/15 08:40	00:02:16	0%	

このステータスはサーバー側のリソースを一部解放します。これに加え、4Dリモートアプリケーションはスリープモードから復帰したときに自動的に 4D Server へと再接続します。

サポートされるシナリオは、以下の様なものです: たとえばお昼休みなどでリモートユーザーが作業を中断するも、サーバーとの接続は開いたままにしたします。マシンはスリープモードへと切り替わります。ユーザーが戻ってきてマシンをスリープから復帰させると、4Dリモートアプリケーションは自動的にサーバーへの接続を復元するとともにセッションコンテキストも復元します。

スリープ状態のリモートセッションは、48時間活動しないとサーバーから自動的に切断されます。このデフォルトのタイムアウトを変更するには、`SET DATABASE PARAMETER` コマンドの `Remote connection sleep timeout` セレクターを使用します。

検索/フィルターエリア

この機能を使用して、検索エリアに入力されたテキストに対応する行だけをリストに表示させ、行数を減らすことができます。エリアには、どの列に対して検索/フィルターが実行されるかが表示されています。ユーザーページでは、4D ユーザー、マシン名、そしてセッション名です。

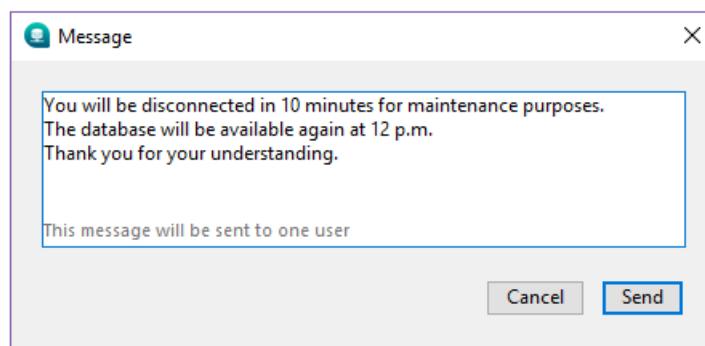
エリアにテキストが入力されると、リストはリアルタイムで更新されます。値をセミコロンで区切ることで、一つ以上の値を使用して検索をおこなうことができます。この場合 OR タイプの演算がおこなわれます。たとえば、"John;Mary;Peter" と入力すると、John または Mary または Peter が対象となる列にある行のみが表示されます。

管理ボタン

このページには 3つのコントロールボタンがあります。これらのボタンは、最低 1つの行が選択されているときに有効になります。 Shiftキーを押しながらクリックして連続した行を、あるいは Ctrl (Windows) / Command (macOS) キーを押しながらクリックして連続しない行を複数選択できます。

メッセージ送信

このボタンを使用して、ウインドウで選択した 4D ユーザーにメッセージを送信できます。ユーザーが選択されていないと、ボタンを使用できません。ボタンをクリックするとダイアログボックスが表示され、メッセージを入力できます。ダイアログにはメッセージを受信するユーザーの数が表示されます：



クライアントマシン上でこのメッセージは警告メッセージとして表示されます。

SEND MESSAGE TO REMOTE USER コマンドを使用することでも、リモートユーザーに対して同じアクションを実行することができます。

プロセス監視

このボタンをクリックすると、選択されたユーザーのプロセスを、管理ウインドウの [プロセス ページ](#) に直接表示させることができます。ボタンをクリックすると、4D Server はプロセスページに移動し、このページの検索/フィルターエリアに選択されたユーザー名を入力します。

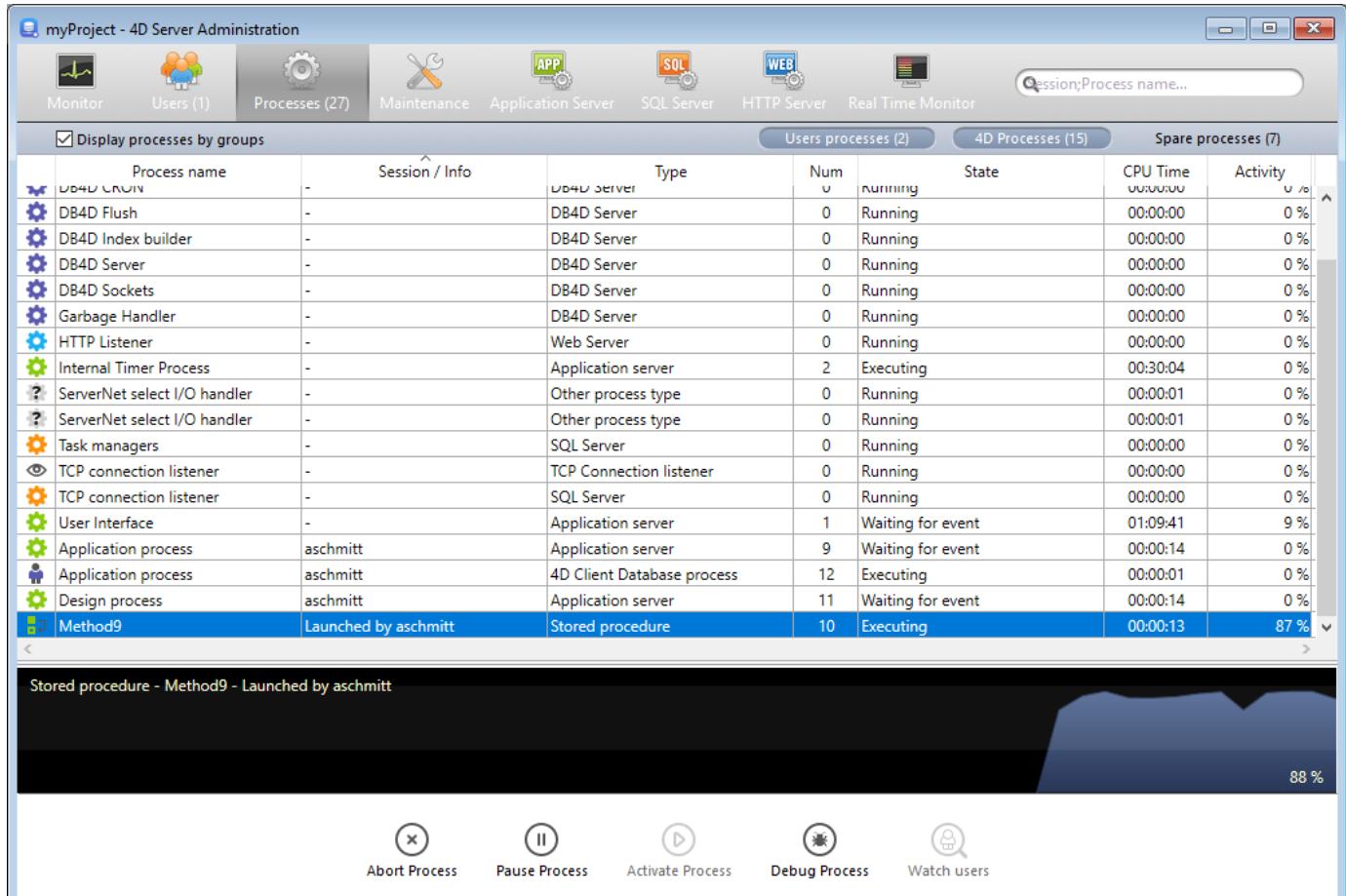
ユーザーをドロップ

このボタンは、選択したユーザーの接続を強制的に解除するために使用します。このボタンをクリックすると警告ダイアログが表示され、接続解除を実行するかキャンセルするか選択できます。確認ダイアログなしに選択ユーザーの接続を解除するには、Altキーを押しながら、ユーザーをドロップ ボタンをクリックします。

DROP REMOTE USER コマンドを使用することでも、リモートユーザーに対して同じアクションを実行することができます。

プロセスページ

プロセスページには実行中のプロセスが表示されます。



The screenshot shows the 'myProject - 4D Server Administration' window with the 'Processes (27)' tab selected. The main area displays a table of processes with columns: Process name, Session / Info, Type, Num, State, CPU Time, and Activity. A checkbox 'Display processes by groups' is checked at the top left. Below the table, a message box titled 'Stored procedure - Method9 - Launched by aschmitt' shows a progress bar at 88%. At the bottom are buttons for Abort Process, Pause Process, Activate Process, Debug Process, and Watch users.

Process name	Session / Info	Type	Num	State	CPU Time	Activity
DB4D Cntriv	-	DB4D Server	0	Running	00:00:00	0 %
DB4D Flush	-	DB4D Server	0	Running	00:00:00	0 %
DB4D Index builder	-	DB4D Server	0	Running	00:00:00	0 %
DB4D Server	-	DB4D Server	0	Running	00:00:00	0 %
DB4D Sockets	-	DB4D Server	0	Running	00:00:00	0 %
Garbage Handler	-	DB4D Server	0	Running	00:00:00	0 %
HTTP Listener	-	Web Server	0	Running	00:00:00	0 %
Internal Timer Process	-	Application server	2	Executing	00:30:04	0 %
ServerNet select I/O handler	-	Other process type	0	Running	00:00:01	0 %
ServerNet select I/O handler	-	Other process type	0	Running	00:00:01	0 %
Task managers	-	SQL Server	0	Running	00:00:00	0 %
TCP connection listener	-	TCP Connection listener	0	Running	00:00:00	0 %
TCP connection listener	-	SQL Server	0	Running	00:00:00	0 %
User Interface	-	Application server	1	Waiting for event	01:09:41	9 %
Application process	aschmitt	Application server	9	Waiting for event	00:00:14	0 %
Application process	aschmitt	4D Client Database process	12	Executing	00:00:01	0 %
Design process	aschmitt	Application server	11	Waiting for event	00:00:14	0 %
Method9	Launched by aschmitt	Stored procedure	10	Executing	00:00:13	87 %

"プロセス" ボタンには、サーバーで実行中のプロセス数が括弧内に表示されます（この番号は、ウィンドウに適用される表示フィルターや グループ毎にプロセスを表示 オプションのステータスを考慮しません）。

列ヘッダーをドラッグ & ドロップして、列の順番を入れ替えることができます。また、ヘッダーをクリックすると、リストの値が並べ替えられます。

ユーザーページと同様にこのページにも、検索欄に入力されたテキストに対応する行だけをリストに表示させ、行数を減らすことができる動的な [検索/フィルターエリア](#) があります。検索/フィルターはセッションとプロセス名の列に対して実行されます。

ウィンドウに表示されるプロセスを、タイプ毎にフィルターするためのボタンが 3つあります：



- ユーザープロセス: ユーザーセッションにより、またユーザーセッションのために作成されたプロセス。このプロセスには人のアイコンが表示されます。
- 4D プロセス: 4D Server エンジンが生成したプロセス。このプロセスには歯車のアイコンが表示されます。
- 予備プロセス: 使用されていないが一時的に保持され、いつでも再利用が可能なプロセス。このメカニズムは 4D Server の反応性を向上させます。このプロセスには薄暗い人のアイコンが表示されます。

グループ毎にプロセスを表示 オプションを使用して、4D Server の内部プロセスやクライアントプロセスをグループ化できます。このオプションをチェックすると：

- 4Dクライアントのプロセス（メインの 4Dクライアントプロセスや 4Dクライアントの基本プロセス。 [プロセスタイプ](#) 参照）は 1つにグループ化されます。
- "タスクマネージャー" グループが作成され、タスクを分割するための内部プロセス（共有バランサー、ネットセッションマネージャー、Exclusive pool worker）がグループ化されます。
- "クライアントマネージャー" グループが作成され、これにはクライアントのさまざまな内部プロセスが含まれます。

ウィンドウの下段には選択したプロセスの稼働状況がグラフィカルに表示されます。

Shiftキーを押しながら連続した行を、Ctrl (Windows) / Command (macOS) キーを押しながら非連続の行を選択できます。

プロセスの稼働状況は、4D Server がこのプロセスのために使用した時間のパーセンテージです。ウィンドウにはプロセスごとに以下の情報が表示されます：

- プロセスタイプ (後述)
- セッション/情報：
 - 4Dプロセス - 空白
 - ユーザープロセス - 4Dユーザー名
 - Webプロセス - URLパス
- プロセス名
- プロセス番号 (たとえば `New process` 関数で返される値)。プロセス番号はサーバー上で割り当てられる番号です。グローバルプロセスの場合、この番号はクライアントマシン上で割り当てられた番号と異なる場合があります。
- プロセスの現在の状況
- 作成されてからのプロセスの実行時間 (秒)
- 4D Server がこのプロセスに使用した時間のパーセンテージ

プロセスタイプ

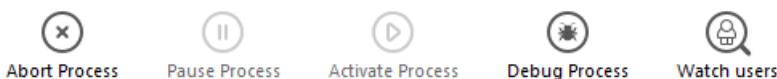
プロセスタイプはアイコンで識別できます。アイコンの色や形に対応するプロセスタイプは以下のとおりです：

icon	type
	アプリケーションサーバー
	SQL サーバー
	DB4D サーバー (データベースエンジン)
	Web サーバー
	SOAP サーバー
	保護された 4Dクライアントプロセス (接続された 4D の開発プロセス)
	メイン4Dクライアントプロセス (接続された 4D のメインプロセス。クライアントマシン上で作成されたプロセスに対応するサーバープロセス)
	4Dクライアント基本プロセス (4Dクライアントプロセスと並列なプロセス。メイン4Dクライアントプロセスをコントロールするプリエンプティブプロセス)
	予備プロセス (以前または後の "4Dクライアントデータベースプロセス")
	SQL サーバーワーカープロセス
	HTTP サーバーワーカープロセス
	4Dクライアントプロセス (接続された 4D 上で実行中のプロセス)
	ストアドプロシージャー (接続された 4D により起動され、サーバー上で実行しているプロセス)
	Web メソッド (4DACTION などにより起動)
	Web メソッド (プリエンプティブ)
	SOAP メソッド (Webサービスにより起動)
	SOAP メソッド (プリエンプティブ)
	ロガー
	TCP接続リスナー
	TCPセッションマネージャー
	その他のプロセス
	ワーカープロセス (コオペラティブ)
	4Dクライアントプロセス (プリエンプティブ)
	ストアドプロシージャー (プリエンプティブプロセス)
	ワーカープロセス (プリエンプティブ)

グループ毎にプロセスを表示 オプションがチェックされていると、それぞれの 4Dクライアントメインプロセスと、その対である 4Dクライアント基本プロセスは一緒にグループ化されて表示されます。

管理ボタン

このページには、選択されたプロセスに対して動作する 5つのコントロールボタンがあります。ユーザープロセスに対してのみ使用できる点に注意してください。



- プロセスを中断: 選択したプロセスをアボートします。このボタンをクリックすると警告ダイアログが表示され、操作を続行またはキャンセルできます。

確認ダイアログなしに選択したプロセスをアボートするには、Altキーを押しながらこのボタンをクリックするか、`ABORT PROCESS BY ID` コマンドを使用します。

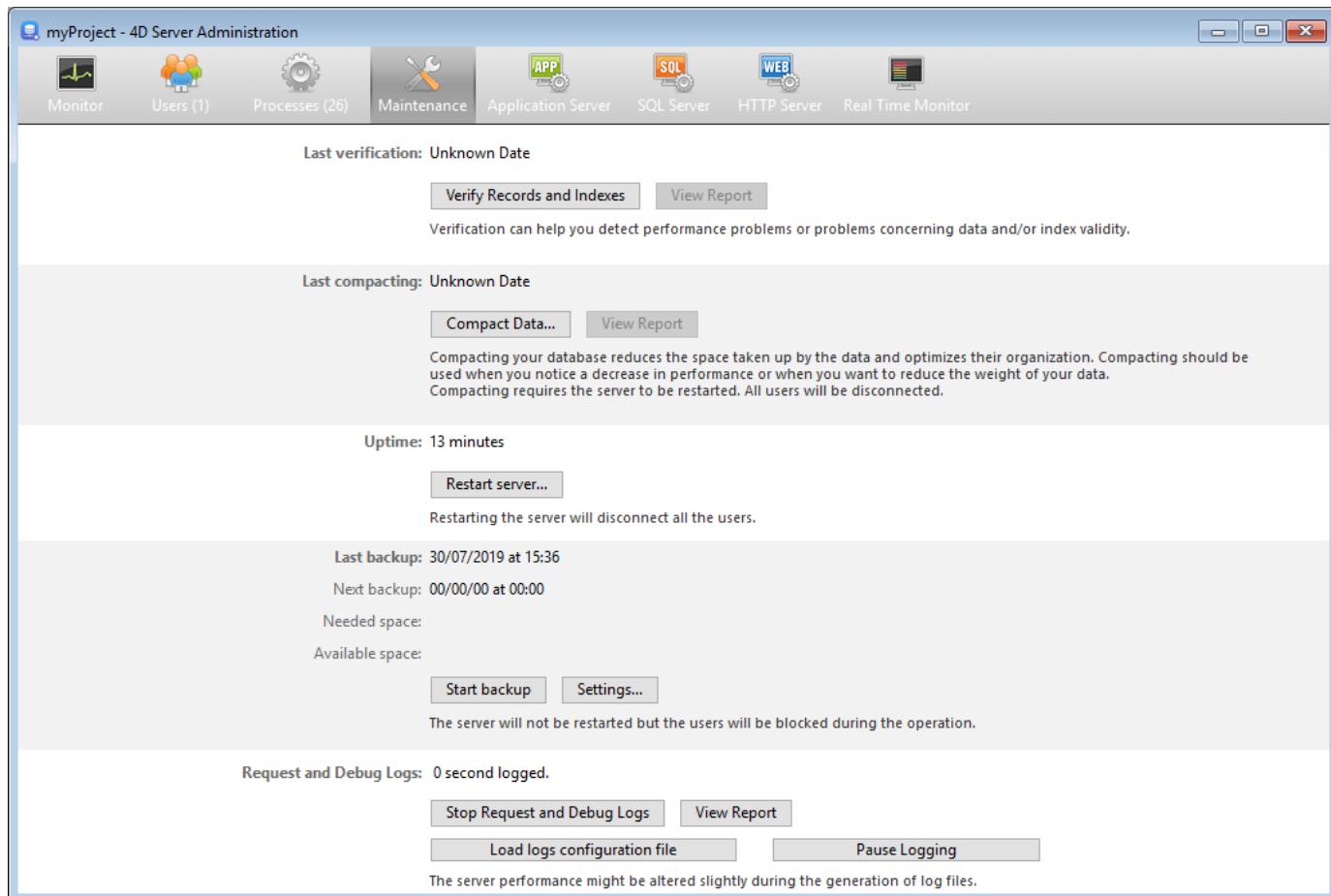
- プロセスを一時停止: 選択したプロセスを一時停止します。
- プロセスをアクティベート: 選択したプロセスの実行を再開します。対象のプロセスは、前述のボタンかプログラムにより一時停止状態でなければなりません。そうでなければ、このボタンは効果ありません。
- プロセスをデバッグ: 選択したプロセスのデバッガーをサーバーマシン上で開きます。このボタンをクリックすると警告ダイアログが表示され、操作を続行またはキャンセルできます。4Dコードが実際にサーバーマシン上で実行されている場合にのみ、デバッガーウィンドウが表示される点に注意してください(たとえば、トリガーや "サーバー上で実行" 属性を持つメソッドの実行時など)。

確認ダイアログなしに選択したプロセスをデバッグするには、Altキーを押しながらこのボタンをクリックします。

- ユーザーを表示: 選択されたプロセスのユーザーを管理ウィンドウの [ユーザーページ](#) に直接表示させることができます。1つ以上のユーザー/プロセスが選択されている場合にこのボタンは有効になります。

メンテナンスページ

メンテナンス ページには、アプリケーションの現在の動作状況に関する情報が表示されます。また、基本的なメンテナンス機能にアクセスすることもできます：



最新の検査/圧縮:

このエリアには、データベース上で実行された最新の [データの検査](#) および [圧縮処理](#) の日付、時刻、状況が表示されます。

レコードとインデックスを検査

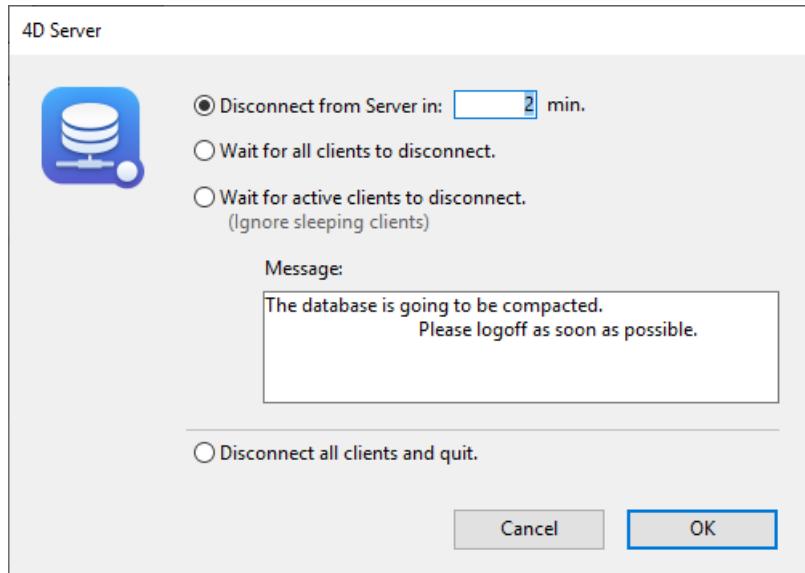
このボタンを使用して、サーバーを止めることなく検査処理を直接起動できます。検証の間、サーバーの動作が遅くなるかもしれないことに留意してください。

データベースのすべてのレコードとすべてのインデックスが検証されます。検査対象を絞り込んだり、追加のオプションを指定したい場合は、[Maintenance & Security Center \(MSC\)](#) を使用します。

検査後、サーバー上の [maintenance Logs](#) フォルダーに、XML形式でレポートファイルが作成されます。レポートを表示（クライアントマシンから処理が実行された場合は レポートをダウンロード）ボタンをクリックすると、ブラウザにレポートを表示できます。

データ圧縮

このボタンを使用して、データ圧縮処理を直接起動できます。この処理をおこなうにはサーバーを停止させる必要があります。ボタンをクリックすると、4D Server の終了ダイアログが表示され、終了方法を選択することができます：



アプリケーションが実際に停止された後、4D Server はデータベースのデータに対する標準の圧縮処理をおこないます。追加のオプションを指定したい場合は、[Maintenance & Security Center \(MSC\)](#) を使用します。

圧縮が終了すると、4D Server は自動でデータベースを再開します。その後、4Dユーザーの再接続が可能になります。

圧縮リクエストがリモートの 4Dマシンからなされた場合、このマシンは 4D Server により自動で再接続されます。

検査後、サーバー上の [maintenance Logs](#) フォルダーに、XML形式でレポートファイルが作成されます。レポートを表示（クライアントマシンから処理が実行された場合は レポートをダウンロード）ボタンをクリックすると、ブラウザーにレポートを表示できます。

動作時間

このエリアには、サーバーが開始されてからの稼働時間（日、時、分）が表示されます。

サーバを再起動…

このボタンをクリックするとサーバーを即座に再起動できます。ボタンをクリックすると、4D Server の終了ダイアログが表示され、終了方法を選択することができます。再起動後、4D Server は自動でプロジェクトを再度開きます。その後、4Dユーザーの再接続が可能になります。

再起動リクエストがリモートの 4Dマシンからなされた場合、このマシンは 4D Server により自動で再接続されます。

前回のバックアップ

このエリアには、データベースの [前回のバックアップ](#) の日付・時刻と、予定された次のバックアップがあれば、それに関する情報が表示されます。自動バックアップは、ストラクチャー設定の スケジューラー ページで設定します

- 前回のバックアップ: 前回のバックアップ日時。
- 次回のバックアップ: 予定された次のバックアップ日時。
- 必要なスペース: バックアップに必要な計算された空き容量。バックアップファイルの実際のサイズは（圧縮などの）設定や、データファイルの変化により変わります。
- 空きスペース: バックアップボリュームの空き容量。

バックアップ開始 ボタンを使用して、現在のバックアップパラメーター（バックアップするファイル、アーカイブの場所、オプションなど）を使用したバックアップを即座に開始できます。環境設定… ボタンをクリックして、これらのパラメーターを確認できます。サーバー上でのバックアップがおこなわれる間、クライアントマシンはロックされ（ただし接続解除はされません）、新規のクライアント接続はできなくなります。

リクエストとデバッグログ

このエリアには、（ログファイルが有効化されている場合に）ログファイルの記録に要した時間が表示され、ログの有効化も管理できます。

ログファイルについては、[ログファイルの詳細](#) を参照ください。

リクエストとデバッグのログを開始/停止

リクエストとデバッグのログを開始 ボタンでログファイルが開始されます。これによりフォーマンスが著しく低下する場合があるため、これはアプリケーションの開発フェーズでのみ使用します。

このボタンは、サーバー上で実行されているオペレーションしか記録しません。

ログが有効になると、ボタンのタイトルが **リクエストとデバッグのログを停止** に変わり、いつでもリクエストの記録を停止できます。停止後にログを再開すると、以前のファイルは消去されることに留意してください。

レポートを表示

レポートを表示（リモートのデスクトップクライアントから処理を実行した場合は [レポートをダウンロード](#)）ボタンをクリックすると、システムウインドウが開いて、リクエストログファイルが表示されます。

ログ設定ファイルを読み込む

このボタンを押すと、特殊な [ログ設定ファイル](#)（.json ファイル）をサーバーに読み込むことができます。この特殊ファイルは、特定のケースを監視・調査するために 4Dテクニカルサービスが提供することができます。

ログを停止する

このボタンを押すと、サーバーで開始されたログを記録する処理がすべて中断されます。この機能は、サーバーの処理を一時的に軽くするのに便利です。

ログが停止されると、ボタンのタイトルが **ログを再開する** に変わり、これを押すことでログの記録を再開することができます。

ログの停止や再開は、[SET DATABASE PARAMETER](#) コマンドでおこなうこともできます。

アプリケーションサーバーページ

アプリケーションサーバーページには、4D Server が公開しているデスクトップアプリケーションについての情報がまとめられていて、公開を管理できます。

The screenshot shows the 'myProject - 4D Server Administration' window. The top navigation bar includes icons for Monitor, Users (1), Processes (26), Maintenance, Application Server (selected), SQL Server, HTTP Server, and Real Time Monitor. The main content area displays information about the Application Server:

- State:** Started
- Starting time:** 27/07/2021 at 18:29
- Uptime:** 1 day 15 hours 59 minutes
- Reject new connections** button (disabled)
- Configuration** section:
 - Structure file: "myProject.4DProject" in volume "D:"
 - Data file: "data.4DD" in volume "D:"
 - Log file: data.journal
 - Mode: Interpreted
- Launched as service:** No
- Listening to IP:** 192.168.7.47
- Port:** 19813
- TLS enabled:** Yes
- Memory** section:
 - Used cache memory: 733,88 kB
 - Total cache memory: 400 MB
- Application Server Connections** section:
 - Maximum: 2
 - Used: 1

ページの上部には、4D Server アプリケーションサーバーの現在の状況が表示されます。

- 状況: 開始または停止
- 開始時刻: アプリケーションサーバーの起動日と時刻。これは、4D Server によってプロジェクトが開かれた日付です。
- 動作時間: プロジェクトがサーバーによって最後に開かれてからの経過時間。

新規接続を許可/拒否

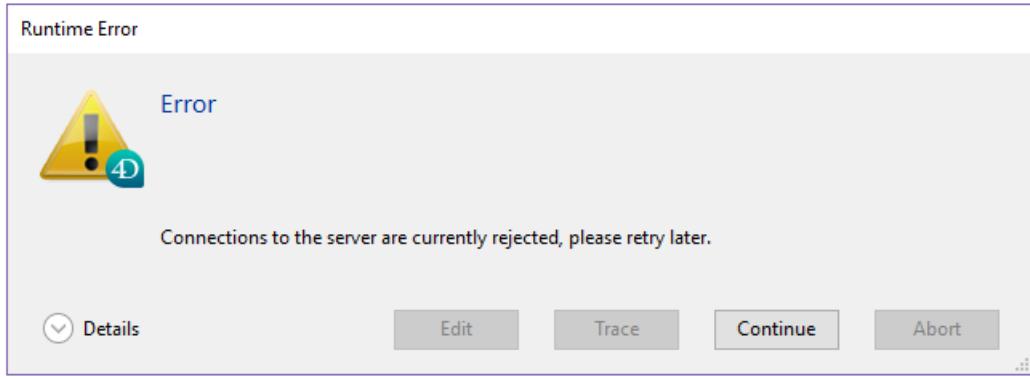
このボタンは切り替え表示され、アプリケーションサーバーへの新しいデスクトップクライアントの接続を管理します。

プロジェクトが公開された時、デフォルトでは:

- ボタンのラベルは "新規接続を拒否" です。
- ライセンスが許可する限り、新規クライアントは自由に接続が可能です。
- プロジェクト名は接続ダイアログに公開されます ("起動時にデータベースを公開する" オプションがストラクチャー設定で有効になっている場合)。

新規接続を拒否 ボタンをクリックすると:

- ボタンのラベルは "新規接続を許可" に変わります。
- 新規クライアントは接続不可になります。接続しようとしたクライアントには以下のメッセージが表示されます:



- 接続ダイアログにプロジェクト名が表示されなくなります。
- すでに接続済みのクライアントは接続解除されず、通常通りに操作が可能です。

REJECT NEW REMOTE CONNECTIONS コマンドを使用することでも、同じアクションを実行することができます。

- 新規接続を許可 ボタンをクリックすると、アプリケーションサーバーはデフォルトに戻ります。

この機能はたとえば、サーバー開始直後に管理者が様々なメンテナンス操作（検証や圧縮など）をおこなうことを可能にします。管理者がクライアント接続を使用する場合、この機能により一つのクライアントだけがデータを更新できることを確実にできます。また、クライアントマシンが接続されていない状態でおこなわなければならないメンテナンス操作の準備のために、この機能を使用することができます。

情報

設定

このエリアには、サーバーが公開する 4Dプロジェクトについての情報（データやストラクチャーファイルの名称と場所、データベースログファイルの名称）が表示されます。ストラクチャーやデータファイル名をクリックすると、完全なパス名を表示させることができます。

モード 欄はアプリケーションの現在の実行モード、コンパイル済みかインタープリターかが表示されます。

エリアの下部には、サーバー設定パラメーター（サービスとして起動、ポート、IP アドレス）や、クライアント/サーバー接続用の TSL の状態（SQL や HTTP 接続は別設定）が表示されます。

メモリ

このエリアには、総キャッシュメモリ（ストラクチャー設定で設定されたパラメーター）と 使用キャッシュメモリ（必要に応じて 4D Server が動的に割り当て）が表示されます。

アプリケーションサーバー接続数

- 最高：アプリケーションサーバーに許可された最大のクライアント同時接続数を表します。この値は、サーバーマシンにインストールされているライセンスによります。
- 使用中：現在使用中の実際の接続数を表します。

SQLサーバーページ

SQLサーバーページには、4D Server に統合された SQLサーバーについての情報が集められています。また、SQLサーバーを有効にするためのボタンも含まれています。

The screenshot shows the 'myProject - 4D Server Administration' window. The top navigation bar includes icons for Monitor, Users (1), Processes (26), Maintenance, Application Server, SQL Server (selected), HTTP Server, and Real Time Monitor. The main content area displays the following information for the SQL Server:

- State:** Started
- Starting time:** 29/07/2021 at 10:30
- Uptime:** Less than one minute
- Stop SQL Server** button (disabled)

Configuration section details:

- Auto-launched at startup: No
- Listening to IP: 192.168.7.47
- Listening on port: 19812
- TLS enabled: No

Connections section shows:

- Number of connections: 0

Maximum connections section shows:

- SQL Server: None

ページの上部には、4D Server の SQLサーバーの現在の状況が表示されます。

- 状況: 開始または停止
- 開始時刻: SQLサーバーの起動日と時刻。
- 動作時間: SQLサーバーが最後に開始されてからの経過時間。

SQLサーバー開始/停止

このボタンは切り替え表示され、4D Server SQLサーバーをコントロールするために使用します。

- SQLサーバーの状態が "開始" の場合、ボタンのタイトルは SQLサーバー停止 になります。このボタンをクリックすると、4D Server SQLサーバーは即座に停止し、指定した TCPポートで受信される外部からの SQLクエリには応答しなくなります。
- SQLサーバーの状態が "停止" の場合、ボタンのタイトルは SQLサーバー開始 になります。このボタンをクリックすると、4D Server SQLサーバーは即座に開始し、指定した TCPポートで受信される外部からの SQLクエリに応答します。4D SQLサーバーを使用するには、適切なライセンスが必要な点に注意してください。

ストラクチャー設定で設定してアプリケーション起動と同時に、またはプログラムを使用して必要な時に、SQLサーバーを自動で開始することができます。

情報

設定

このエリアには、SQLサーバー設定のパラメーターが表示されます：開始時の自動起動、待受IPアドレス、待受TCPポート（デフォルトで 19812）、そして SQL接続用の TLS の状態（4D や Web接続は別設定）。

これらの値は 4D のストラクチャー設定で変更できます。

接続

4D Server上で現在開かれている SQL接続の数。

最大接続数

許可される同時SQL接続の最大数。この値は、サーバーマシンにインストールされているライセンスによります。

HTTPサーバーページ

HTTPサーバー ページには、4D Server の Webサーバーや SOAPサーバーに関する情報が集められています。Webサーバーは、HTMLページやピクチャーなどの Webコンテンツの公開を可能にします。SOAPサーバーは Webサービスの公開を管理します。これら 2つのサーバーは、4D Server の内部的な HTTPサーバーに依存しています。

The screenshot shows the 'myProject - 4D Server Administration' window. The top navigation bar includes icons for Monitor, Users (1), Processes (26), Maintenance, Application Server, SQL Server, **HTTP Server** (selected), and Real Time Monitor. The main content area is divided into several sections:

- HTTP Server**: State: Started, Starting time: 27/07/2021 at 18:29, Uptime: 1 day 16 hours 3 minutes, Total HTTP hits: 0. Includes a 'Stop HTTP server' button.
- Web information**: Web requests: Accepted, Maximum Connections: None.
- SOAP information**: SOAP requests: Accepted, Maximum Connections: None. Includes a 'Reject SOAP requests' button.
- HTTP server Configuration**: Auto-launched at startup: Yes, HTTP Server processes (used/total): 5/7, Cache memory: 0 pages (0%). Includes a 'Clear cache' button. Sub-sections include Listening to IP: 192.168.7.47, HTTP port: 80, TLS enabled: Yes, HTTPS Port: 443, Log file: Logweb.txt, Log format: -, and Next log backup: 00/00/00 at 00:00.

ページの上部には、4D Server の HTTPサーバーの現在の状況が表示されます。

- 状況: 開始または停止
- 開始時刻: HTTPサーバーの起動日と時刻。
- 動作時間: HTTPサーバーが最後に開始されてからの経過時間。
- 総HTTPヒット数: HTTPサーバーが開始されてから、サーバーが受信したローレベルの HTTPヒット数。

HTTPサーバー開始/停止

このボタンは切り替え表示され、4D Server HTTPサーバーをコントロールするために使用します。

- HTTPサーバーの状態が "開始" の場合、ボタンのタイトルは HTTPサーバー停止 になります。このボタンをクリックすると、4D Server HTTPサーバーは即座に停止し、Webサーバー、RESTサーバー、および SOAPサーバーはリクエストを受け付けなくなります。
- HTTPサーバーの状態が "停止" の場合、ボタンのタイトルは HTTPサーバー開始 になります。このボタンをクリックすると、4D Server HTTPサーバーは即座に開始し、Web、REST、および SOAPリクエストを受け付けます。

HTTPサーバーを開始するには適切なライセンスが必要です。

ストラクチャー設定で設定してアプリケーション起動と同時に、またはプログラムを使用して必要な時に、HTTPサーバーを自動で開始することができます。

Web情報

このエリアには、4D Server の Webサーバーに関する情報が表示されます。

- Web リクエスト: 受け入れ、または拒否。この情報は Webサーバーが有効かどうかを示します。Webサーバーは直接 HTTPサーバーにリンクしているため、HTTPサーバーが開始されていれば Webリクエストは受信され、停止されていれば拒否されます。
- 最大接続数: 許可される Web接続の最大数。この値は、サーバーマシンにインストールされているライセンスによります。

SOAP情報

このエリアには、4D Server の SOAPサーバーに関する情報が表示され、コントロールボタンも一つ含まれます。

- SOAP リクエスト: 受け入れ、または拒否。この情報は SOAPサーバーが有効かどうかを示します。SOAPリクエストを受け入れるためにには、HTTPサーバーが開始され、かつ SOAPサーバーが明示的にリクエストを受け入れなければなりません (ボタンの説明参照)。
- 最大接続数: 許可される SOAP接続の最大数。この値は、サーバーマシンにインストールされているライセンスによります。
- SOAPリクエストを受け入れる/受け入れないボタン: このボタンは切り替え表示され、4D Server SOAPサーバーのコントロールに使用します。このボタンをクリックすると、ストラクチャー設定の "Webサービス" ページの Webサービスリクエストを許可するオプションが変更されます。また、ストラクチャー設定の当該オプションが変更されれば、ボタンのラベルも変わります。また、**SOAP REJECT NEW REQUESTS** コマンドを使って新規の SOAPリクエストを拒否することもできますが、このコマンドは Webサービスリクエストを許可するオプションの値を変更しません。

HTTPサーバー停止中に SOAPリクエスト受け入れるボタンをクリックすると、4D は自動で HTTPサーバーを開始します。

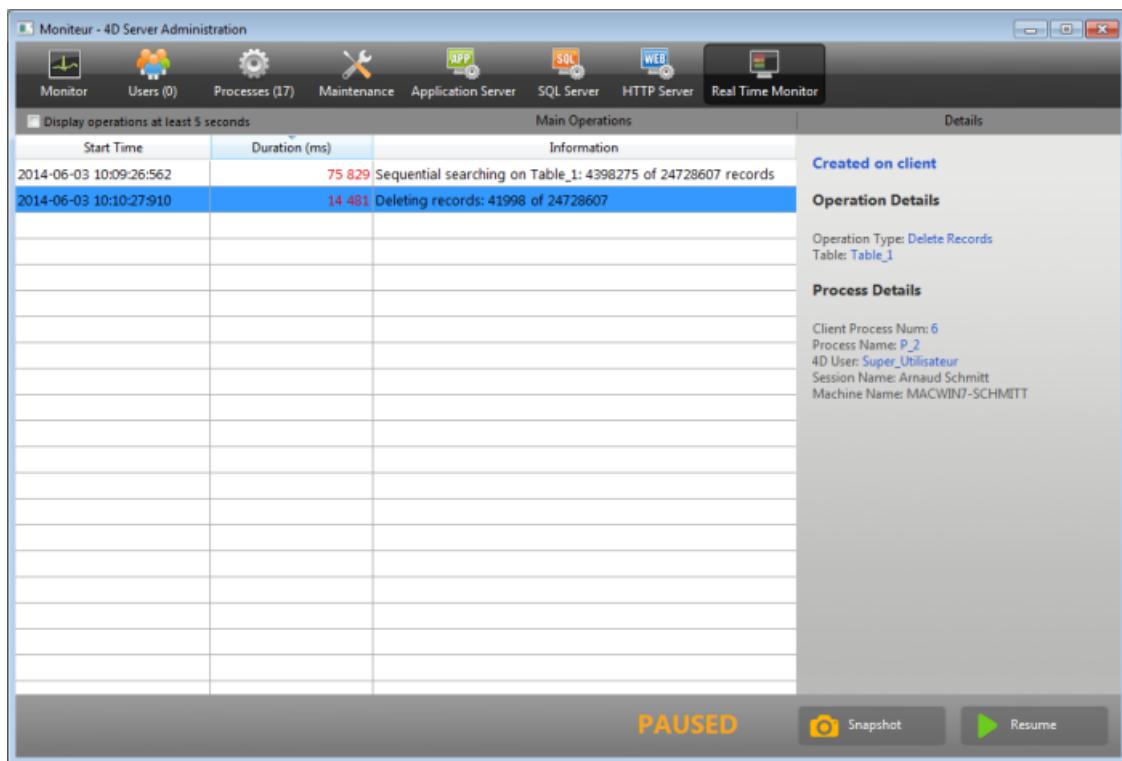
HTTPサーバー設定

このエリアには、HTTPサーバーの設定パラメーターや動作に関する情報が表示されます。

- 開始時に自動起動: ストラクチャー設定で設定されたパラメーター。
- HTTP サーバープロセス (使用/総計): サーバー上で作成されたHTTPプロセス数 (現在のプロセス数 / 作成されたプロセスの総数)。
- キャッシュメモリ: HTTPサーバーキャッシュメモリのサイズ (キャッシュが実際に使用しているサイズ / ストラクチャー設定で理論的にキャッシュに割り当てられた最大サイズ)。キャッシュクリア ボタンをクリックすると、現在のキャッシュを空にすることができます。
- 待受IP、HTTPポート (デフォルトは 80)、HTTP接続用の TSL有効 (4D と SQL接続は別設定)、および HTTPSポート: これらは、ストラクチャー設定またはプログラミングにより設定された、HTTPサーバーの現在の **設定パラメーター** を表示します。
- ログファイル情報: 名称、フォーマット、および HTTPサーバーの次回の自動ログバックアップの日付 (logweb.txt ファイル)。

リアルタイムモニターページ

リアルタイムモニターは、アプリケーションによって実行された、"長い" オペレーションの状態をリアルタイムでモニターします。これらのオペレーションとは、たとえばシーケンシャルクエリやフォーミュラの実行などです:



このページは、サーバーマシンの管理ウィンドウにありますが、リモートの 4Dマシンからも見られます。リモートマシンの場合は、サーバーマシン上で実行されている操作のデータを表示します。

データに対して実行されている長い処理は、それぞれに行が割り当てられます。操作が完了すると、この行は消えます（オペレーションを最低5秒間表示するオプションをチェックすることで、短いオペレーションでも5秒間表示したままにできます。以下参照）。

各行について、以下の情報が表示されます:

- 開始時刻: 操作の開始時刻が、"dd/mm/yyyy - hh:mm:ss" というフォーマットで表示されます。
- 経過時間 (秒): 進行中の操作の経過時間が秒単位で表示されます。
- 情報: 操作の説明。
- 詳細: このエリアには、選択したオペレーションのタイプに応じて、その詳細な情報が表示されます。具体的には以下の情報が表示されます:
 - 作成された場所: そのオペレーションがクライアントアクションの結果か（クライアント上で作成）、ストアドプロシージャーまたは "サーバー上で実行" オプションを使用した結果か（サーバー上で作成）を表示します。
 - オペレーション詳細: オペレーションタイプと、（クエリオペレーションに対しては）クエリプランを表示します。
 - サブオペレーション（あれば）: 選択したオペレーションに従属するオペレーションを表示します（例: 親レコードの前にリレートレコードを削除する）
 - プロセス詳細: テーブル、フィールド、プロセスやクライアントに関する追加情報が表示されます。オペレーションのタイプによって異なります。

リアルタイムモニターページは、`GET ACTIVITY SNAPSHOT` コマンドを内部的に使用しています。詳細については、コマンドの説明を参照ください。

このページは表示後すぐにアクティブになり、恒久的に更新され続けます。ただし、この処理によって、アプリケーションの実行を極端に遅くさせる可能性があることに注意してください。以下の方法を用いて更新を一時的に停止させることができます:

- 停止 ボタンをクリックする
- リストの中をクリックする
- スペースバーを押す

ページを停止させると一時停止のメッセージが表示され、ボタンの表示が 再開 に変わります。モニタリング停止操作と同じ操作をすることでモニタリングを

再開させることができます。

詳細モード

必要であれば、RTMページはオペレーションごとに追加の情報を表示することができます。

オペレーションのアドバンスドモードにアクセスするには、Shiftキーを押しながら、情報を取得したいオペレーションを選択します。すべての閲覧可能な情報が、フィルタリングなしで "プロセス詳細" エリアに表示されます(GET ACTIVITY SNAPSHOT コマンドで返されるものと同じです)。表示される情報は、選択したオペレーションによって異なります。

標準モードで表示される情報の例です:

Start Time	Duration (ms)	Information	
2014-05-27 16:25:56	2 870	Sequential searching on Table_1: 438712 of 15128756 records	Created on client
			Operation Details
			Operation Type: Query Query Plan: "Table_1]Champ_2"="Table_1]Champ_2"
			Process Details
			Client Process Num: 7 Process Name: P_3 4D User: Super_Utilisateur Session Name: Arnaud Schmitt Machine Name: MACWIN7-SCHMITT

アドバンスドモード (オペレーションを Shift+クリック) では、さらなる情報が表示されます:

Start Time	Duration (ms)	Information	
2014-05-27 16:25:56	2 870	Sequential searching on Table_1: 438712 of 15128756 records	Created on client
			Operation Details
			Operation Type: Query Query Plan: "Table_1]Champ_2"="Table_1]Champ_2"
			Process Details
			Client Process Num: 7 Process Name: P_3 4D User: Super_Utilisateur Session Name: Arnaud Schmitt Machine Name: MACWIN7-SCHMITT Client UID: B0C9071B0C9071B0C9071B0C9080C9071 Client Version: v14 R2 Beta

スナップショットボタン

コピー ボタンを使用すると、RTMパネルに表示されている全オペレーションと、それに関連する詳細 (プロセスとサブオペレーション情報) がクリップボードへとコピーされます:



オペレーションを最低5秒間表示する

オペレーションを最低5秒間表示するオプションをチェックすると、表示されたオペレーションはどれも (実行が終了した後も) 最低5秒間は表示されたままになります。このオプションが適用されたオペレーションは、オペレーションリスト中に灰色で表示されます。この機能は、とても早く終わってしまうオペレーションの情報を取得したい場合に有効です。

WebAdmin

WebAdmin とは、4D および 4D Server に使用される組み込みの Webサーバーコンポーネントの名称で、データエクスプローラーなどの管理機能への安全な Webアクセスを提供します。ブラウザーや、任意の Webアプリケーションから、ローカルまたはリモートでこの Webサーバーに接続し、関連の4Dアプリケーションにアクセスすることができます。

WebAdmin 内部コンポーネントは、"WebAdmin" 権限を持つユーザーの認証を処理し、管理セッションを開いて専用インターフェースにアクセスできるようにします。

この機能は、ヘッダレスで動作する 4Dアプリケーションでも、インターフェースを持つ 4Dアプリケーションでも使用できます。

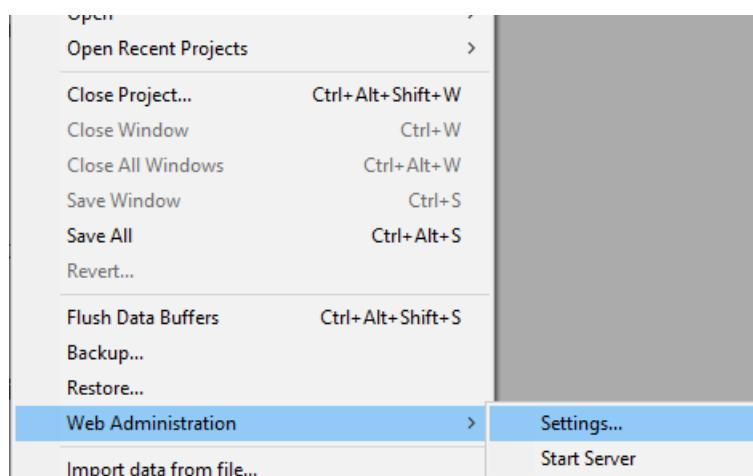
WebAdmin Webサーバーの起動

デフォルトでは、WebAdmin Webサーバーは開始しません。起動時に開始するように設定するか、(インターフェース付きの場合は) メニューから手動で開始する必要があります。

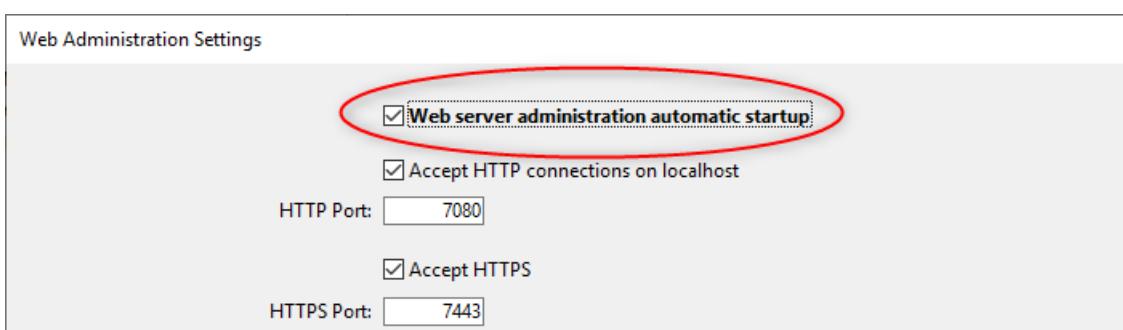
自動スタートアップ

WebAdmin Webサーバーは、4D または 4D Server アプリケーションの起動時 (プロジェクトの読み込み前) に開始するように設定できます。

- インターフェースを持つ 4Dアプリケーションを使用している場合は、ファイル > Web管理 > 設定... メニュー項目を選択します。



Web管理設定ダイアログボックスで、Webサーバー管理自動スタートアップ オプションをチェックします。



- ヘッダレスの 4Dアプリケーションを使用しているかにかかわらず、以下の コマンドライン・インターフェース の引数を使用して、自動スタートアップを有効にすることができます:

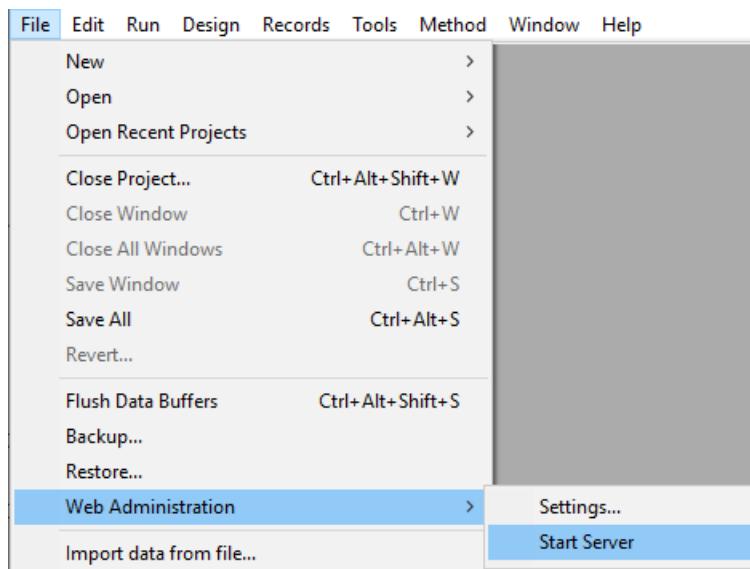
```
open ~/Desktop/4D.app --webadmin=auto-start true
```

WebAdmin Webサーバーが使用する TCPポート (設定により、HTTPS または HTTP) が開始時に空いていない場合、4D は次の 20個のポートを順に試し、利用できる最初のポートを使用します。利用可能なポートがない場合、Webサーバーは開始せず、エラーが表示されるか、(ヘッダレスアプリケーションの場合は) コンソールのログに記録されます。

開始と停止

インターフェースを持つ 4D アプリケーションを使用している場合、プロジェクトの WebAdmin Web サーバーはいつでも開始または停止することができます：

ファイル > Web 管理 > Web サーバー開始 メニュー項目を選択します。



サーバーが開始されていると、メニュー項目は Web サーバー停止 になります。WebAdmin Web サーバーを停止するには、これを選択します。

WebAdmin 設定

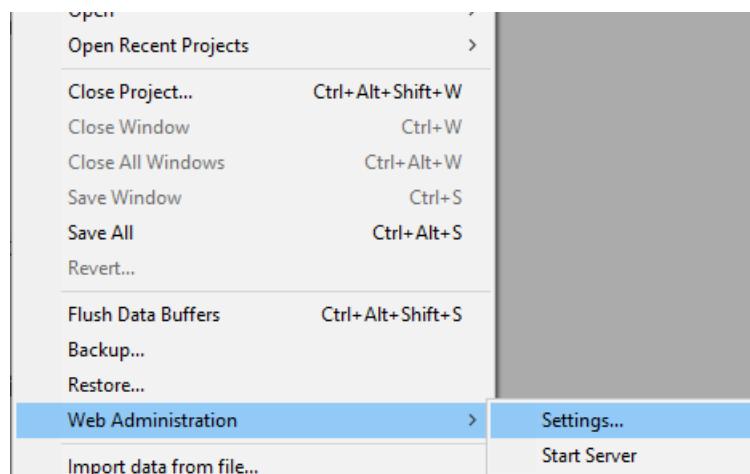
アクセスキーを定義するには、WebAdmin コンポーネントの設定は必須です。デフォルトで、アクセスキーが設定されていない場合は、URL 経由のアクセスは許可されません。

WebAdmin コンポーネントの設定は、[Web 管理設定ダイアログボックス](#)（後述参照）でおこないます。

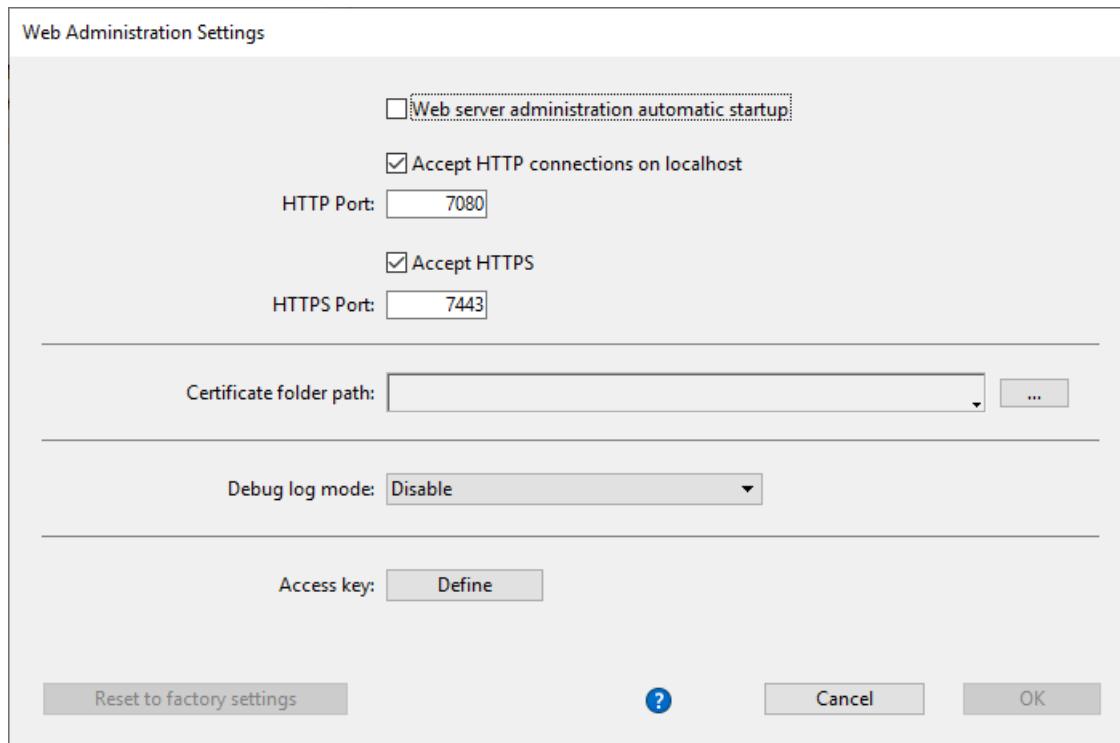
ヘッドレス 4D アプリケーションを使用している場合は、[コマンドライン・インターフェース の引数](#) を使用して基本的な設定を定義できます。高度なパラメーターを定義するには、設定ファイルをカスタマイズする必要があります。

設定ダイアログボックス

Web 管理の設定ダイアログボックスを開くには、ファイル > Web 管理 > 設定... メニュー項目を選択します。



次のようなダイアログボックスが表示されます：



Webサーバー管理自動スタートアップ

4D または 4D Server アプリケーションの起動時に `WebAdmin` Webサーバーを自動的に開始させるには、このオプションをチェックします（[前述参照](#)）。デフォルトでは、このオプションはチェックされていません。

ローカルホストでHTTP接続を受け入れる

このオプションを有効にすると、4Dアプリケーションと同じマシン上で HTTP を介して `WebAdmin` Webサーバーに接続できます。デフォルトでは、このオプションはチェックされています。

注:

- ローカルホスト以外による HTTP接続は受け付けません。
- このオプションがチェックされていても、[HTTPSを受け入れる](#) がチェックされていて、TLS の設定が有効な場合、ローカルホストの接続は HTTPS を使用します。

HTTP ポート

ローカルホストでHTTP接続を受け入れる オプションが有効な場合、`WebAdmin` Webサーバーへの HTTP接続に使用するポート番号です。デフォルト値は 7080 です。

HTTPSを受け入れる

このオプションを有効にすると、`WebAdmin` Webサーバーに HTTPS を介して接続できます。デフォルトでは、このオプションはチェックされています。

HTTPS ポート

HTTPSを受け入れる オプションが有効な場合、`WebAdmin` Webサーバーへの HTTPS接続に使用するポート番号です。デフォルト値は 7443 です。

認証フォルダパス

TLS証明書ファイルが置かれているフォルダーのパスです。デフォルトでは認証フォルダパスは空で、4D または 4D Server は 4Dアプリケーションに組み込まれた証明書ファイルを使用します（カスタム証明書はプロジェクトフォルダーの隣に保存する必要があります）。

デバッグルogモード

HTTPリクエストログファイル（アプリケーションの "Logs" フォルダーに格納されている `HTTPDebugLog_nn.txt` (nn はファイル番号)）の状態やフォーマットを指定します。次のオプションから選択することができます：

- 無効化 (デフォルト)
- bodyパートを全て - レスポンスおよびリクエストのボディパートを含める形で有効化。
- bodyパートを含めない - ボディパートを含めない形で有効化 (ボディサイズは提供されます)
- リクエストのbody - リクエストのボディパートのみを含める形で有効化。
- レスポンスのbody - レスポンスのボディパートのみを含める形で有効化。

アクセスキー

WebAdmin Webサーバーへの URL経由アクセスのロックを解除するには、アクセスキーの定義は必須です (4Dメニュー命令によるアクセスにはアクセスキーは必要ありません)。アクセスキーが定義されていない場合、[データエクスプローラーページ](#)などの Web管理インターフェースに Webクライアントを使って URLを介した接続はできません。接続リクエストがあった場合には、エラーページが返されます:



アクセスキーはパスワードに似ていますが、ログインとは関係ありません。

- 新しいアクセスキーを定義するには、定義 ボタンをクリックし、ダイアログボックスにアクセスキーの文字列を入力して OK をクリックします。すると、ボタンラベルが 編集 に変わります。
- アクセスキーを編集するには、編集 ボタンをクリックし、ダイアログボックスに新しいアクセスキーの文字列を入力して OK をクリックします。
- 新しいアクセスキーを削除するには、編集 ボタンをクリックし、ダイアログボックスのアクセスキー欄を空にして OK をクリックします。

WebAdmin のヘッダレス設定

すべての [WebAdmin 設定](#) は、`WebAdmin.4DSettings` ファイルに保存されます。4D および 4D Server アプリケーション毎にデフォルトの `WebAdmin.4DSettings` ファイルが 1つ存在し、同じホストマシン上で複数のアプリケーションを運用することができます。

4D および 4D Server アプリケーションをヘッダレスで実行している場合、デフォルトの `WebAdmin.4DSettings` ファイルを設定して使用するか、カスタムの `.4DSettings` ファイルを指定することができます。

ファイルの内容を設定するには、インターフェースを持つ 4Dアプリケーションの[WebAdmin設定ダイアログ](#)を使用し、その後ヘッダレスで実行します。このとき、デフォルトの `WebAdmin.4DSettings` ファイルが使用されます。

また、カスタムの `.4DSettings` ファイル (xml形式) を設定し、デフォルトファイルの代わりに使用することもできます。この機能をサポートするために、[コマンドライン・インターフェース](#) ではいくつかの専用の引数が用意されています。

`.4DSettings` ファイルにおいて、アクセスキーは平文では保存されません。

例:

```
"%HOMEPATH%\Desktop\4D Server.exe" MyApp.4DLink --webadmin-access-key
"my Fabulous AccessKey" --webadmin-auto-start true
--webadmin-store-settings
```

認証とセッション

- 事前に本人確認せずに URL経由で Web管理ページにアクセスした場合、認証が必要になります。ユーザーは、認証ダイアログボックスに [アクセスキー](#) を入力する必要があります。 WebAdmin 設定でアクセスキーが定義されていない場合には、URL経由のアクセスはできません。
- 4D または 4D Server のメニュー項目 (レコード > データエクスプローラー または ウィンドウ > データエクスプローラー (4D Server) など) から Web管理ページに直接アクセスした場合、アクセスは認証なしで許可され、ユーザーは自動的に認証されます。

アクセスが許可されると、4Dアプリケーション上に "WebAdmin" 権限を持つ Web セッション が作成されます。カレントセッションが "WebAdmin" 権限を持っている限り、WebAdmin コンポーネントは要求されたページを提供します。

Webデータエクスプローラー

プレビュー: Webデータエクスプローラーは、レビュー機能として提供されています。運用環境でこの機能を使用することは推奨されません。最終的な実装は若干異なる可能性があります。

データエクスプローラーは、プロジェクトのデータストアにあるデータを表示・クリアするための Webインターフェースを提供します。このツールを使用すると、すべてのエンティティを簡単に照会し、属性値に基づいて検索・並べ替え・フィルターすることができます。このツールは、開発プロセスのどの段階においても、データを管理し、問題を迅速に特定するのに役立ちます。

ID	firstname	lastname	salary	birthdate
15	Hermance	ELNOJAS	37 809	19/10/1996
26	Hermelin	SAGELAN	62 409	25/04/1975
41	Helixane	RAPPART	30 709	04/04/1971
65	Harrison	CHESSIT	78 109	17/07/1979
84	Hatrice	MERHEDIUR	62 109	06/04/1996
121	Heleazar	DROENT	74 209	02/10/1970
125	Harn	CIURSOAR	52 209	20/11/1985
139	Hectoria	BANIOT	33 809	09/07/1980
200	Harmonie	CESILO	72 509	11/02/1995
258	Harkaitz	MARZIUG	57 809	13/01/1980
271	Haiza	FEUCHA	59 509	10/03/1970
297	Helma	GARBAL	73 709	03/03/1975
372	Héliena	ERASTA	45 709	15/01/1976

アクセス設定

データエクスプローラーの設定や認証は [WebAdmin](#) Webサーバーコンポーネントに依存しています。

- 設定: データエクスプローラーの設定は、[WebAdmin](#) Webサーバーの設定を再利用します。
- 認証: データエクスプローラーへのアクセスは、認証されたセッションユーザーが、"WebAdmin" 権限を持っている場合に許可されます。データエクスプローラーのメニュー項目（後述参照）からデータエクスプローラーにアクセスした場合、認証は自動的におこなわれます。

データエクスプローラーへのアクセスは、[`.setAdminProtection\(\)`](#) 関数を使って無効化できます。

データエクスプローラーを開く

[WebAdmin](#) Webサーバーを開始すると、データエクスプローラーのページが自動的に利用可能になります。

データエクスプローラーWebページに接続するには:

- インターフェースを持つ 4D アプリケーションを使用している場合は、データエクスプローラー... を次のメニュー内から選択します:
 - レコード メニュー (4D スタンドアロンの場合)
 - ウィンドウ メニュー (4D Server の場合)

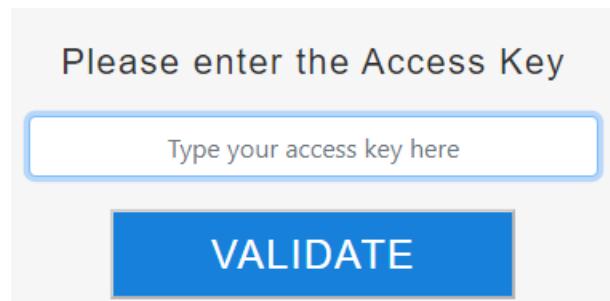
- ヘッダレス4Dアプリケーションを使用しているかどうかに関わらず、Webブラウザを開いて次のアドレスを入力します：

IPaddress:HTTPPort/dataexplorer

または

IPaddress:HTTPSPort/dataexplorer

このコンテキストでは、サーバー上で **WebAdmin** セッションを開くために [アクセスキー](#) の入力を求められます。



HTTPPort および **HTTPSPort** の値は、**WebAdmin** 設定内で定義されます。

データエクスプローラーの使用

データエクスプローラーでは、包括的でカスタマイズ可能なデータの表示に加えて、データの照会や並べ替えをおこなうことができます。

要件

データエクスプローラーは、以下の Webブラウザーをサポートしています。

- Chrome
- Safari
- Edge
- FireFox

データエクスプローラーを使用するための最小解像度は 1280x720 です。推奨解像度は 1920x1080 です。

画面の説明

データエクスプローラーは、[ORDAマッピングルール](#) に基づいて、ORDAデータモデルへの全体的なアクセスを提供します。

ページ下部のセレクターを使って、表示テーマを **ダークモード** に切り替えることができます。

ID	Firstname	Lastname	salary	birthdate
15	Hermann	ELGLORAS	37 809	16/10/1996
26	Hermann	SAGELAN	62 409	25/04/1975
41	Hilma	RAPPART	39 709	04/04/1971
63	Hilma	CHESSIT	78 109	17/03/1979
84	Hilma	MERREDUR	62 109	06/04/1996
121	Hilma	DRENT	74 209	02/10/1970
125	Hilma	CHURGAR	52 209	20/11/1985
139	Hilma	BANOT	33 809	09/07/1980
200	Hilma	CESIL	72 509	11/02/1995
258	Hilma	MARZUS	67 809	13/01/1980
271	Hilma	PEUCHA	59 509	10/01/1970
297	Hilma	GARBAL	79 709	03/03/1975
372	Hilma	GRASTA	45 709	18/01/1978

このページにはいくつかのエリアがあります：

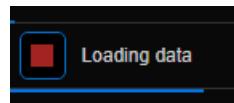
- 左側には **データクラスエリア** と **属性エリア** があり、表示するデータクラスおよび属性を選択できます。属性は、ストラクチャーにおける作成順にしたがって並べられます。プライマリキーおよびインデックス付きの属性には、専用アイコンが表示されます。表示されているデータクラス名と属性名のリストは、それぞれの検索エリアを使ってフィルターできます。

The screenshot shows the 'DataClasses' interface. At the top is a search bar with a magnifying glass icon. Below it is a list of selected entities: 'Employee' and 'events'. The background is dark.

- 中央部には、検索エリアと データグリッド（選択されたデータクラスのエンティティのリスト）があります。グリッドの各列は、データストアの属性を表します。
 - デフォルトでは、すべてのエンティティが表示されます。検索エリアを使用して、表示されるエンティティをフィルターできます。2つのクエリモードがあります： **属性に基づくクエリ**（デフォルト）、および **式による高度なクエリ** です。対応するボタンをクリックして、クエリモードを選択します（X ボタンは、クエリエリアをリセットして、フィルターを停止します）。



- 選択されたデータクラスの名前は、データグリッドの上にタブとして追加されます。これらのタブを使って、選択されたデータクラスを切り替えることができます。参照されているデータクラスを削除するには、データクラス名の右に表示される "削除" アイコンをクリックします。
- 左側の属性のチェックを外すことで、表示されている列数を減らせます。また、ドラッグ & ドロップでデータグリッドの列の位置を入れ替えることができます。列のヘッダーをクリックすると、値に応じて **エンティティを並べ替え** ることができます（可能な場合）。
- 処理に時間がかかる場合は、進捗バーが表示されます。赤いボタンをクリックすると、いつでも実行中の処理を停止できます：



- 右側には 詳細エリアがあり、選択されているエンティティの属性値が表示されます。ピクチャーやオブジェクト（json で表現）を含めた、すべての属性タイプが表示されます。エリア下部にある First / Previous / Next / Last のリンクをクリックすることで、データクラスのエンティティ間を移動することができます。

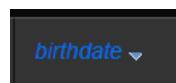
データの更新

データベース側で、ORDAモデルやデータが変更された場合（テーブルの追加、レコードの編集・削除など）、F5キーなどでデータエクスプローラーのページをブラウザーで更新するだけで、変更が反映されます。

エンティティの並べ替え

表示されているエンティティのリストを、属性値に応じて並べ替えることができます。並べ替えには、ピクチャーとオブジェクトを除くすべての属性を使用できます。

- 列のヘッダーをクリックすると、その列の属性値に応じてエンティティを並べ替えます。デフォルトでは、昇順でソートされます。2回クリックすると、降順でソートされます。並べ替えの基準となる列には小さなアイコンが付き、属性名が イタリック で表示されます。



- 属性を基準に複数のレベルでソートできます。たとえば、従業員を都市別にソートした後、給与別にソートすることができます。これには、Shift キーを押しながら、ソート基準とする各列のヘッダーを順にクリックします。

属性に基づくクエリ

このモードでは、データグリッドの属性名の上のエリアに検索（または除外）する値を入力して、エンティティをフィルターします。1つまたは複数の属性でフィルター可能です。入力すると、エンティティリストは自動的に更新されます。

ID	firstname
1	Florette
29	Flora
1 200	Floria

複数の属性を指定した場合は、自動的に AND が適用されます。たとえば次のフィルターでは、*firstname* 属性が "flo" で始まり、*salary* 属性値が > 50000 であるエンティティが表示されます：

firstname	salary
Flora	76 109
Floria	58 009
Florent	79 409

X ボタンは入力された属性値を削除し、フィルターを停止します。

属性のデータ型に応じて、さまざまな演算子やクエリオプションが利用できます。

ピクチャーやオブジェクト属性はフィルターできません。

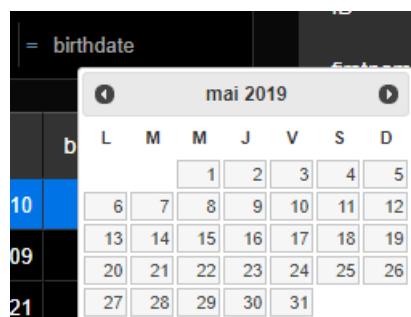
比較演算子

数値、日付、時間型の属性では、デフォルトで "=" 演算子が選択されています。ただし、演算子のリストから別の演算子を選択することができます ("=" アイコンをクリックするとリストが表示されます)。

=	salary
<	
≤	
>	
≥	
≠	

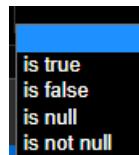
日付

日付型属性では、ピッカーを使って日付を入力することができます（日付エリアをクリックするとカレンダーが表示されます）。



ブール

ブール型の属性エリアをクリックすると、true/false 値だけでなく null/not null 値でもフィルターすることができます。



- null は、その属性値が定義されていないことを示します。
- not null は、属性値が定義されていることを示します（つまり、true または false）。

テキスト

テキストフィルターは、文字の大小を区別しません (a = A)。

フィルターは "～で始まる" タイプです。たとえば、"Jim" と入力すると、"Jim" と "Jimmy" が表示されます。

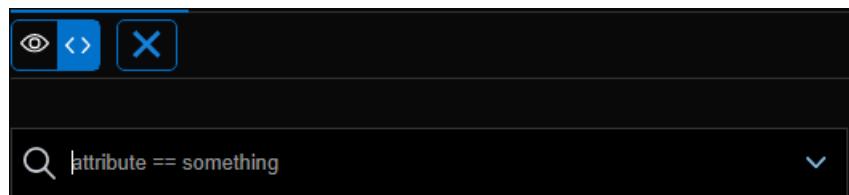
ワイルドカード文字 (@) を使って、1つ以上の開始文字を置き換えることもできます。たとえば：

フィルター文字列	検出結果
Bel	"Bel" で始まるすべての値
@do	"do" を含むすべての値
Bel@do	"Bel" で始まり、"do" を含むすべての値

"完全一致" のような、より詳細なクエリを作成するには、高度なクエリ機能を使用する必要があります。

式による高度なクエリ

このオプションを選択すると、エンティティリストの上にクエリエリアが表示され、コンテンツのフィルターに使用する任意の式を入力することができます。



属性クエリでは利用できない高度なクエリを入力することができます。たとえば、firstname 属性に "Jim" が含まれ、"Jimmy" が含まれていないエンティティを探すには、次のように記述します：

```
firstname=="Jim"
```

[query\(\)](#) 関数の説明 にある ORDA クエリ式を利用することができますが、次の制限および相違点があります：

- セキュリティ上、eval() を使った式を実行することはできません。
- プレースホルダーは使用できません。値の指定された クエリ文字列 を使用しなくてはなりません。
- スペース文字を含む文字列値は、二重引用符 ("") で囲む必要があります。

たとえば、Employee データクラスの場合に、次のように記述できます：

```
firstname = "Marie Sophie" AND manager.lastname = "@th"
```

[queryPlan](#) と [queryPath](#) を両方表示するには v アイコンをクリックします。このエリアでは、サブクエリのブロックにカーソルを合わせると、サブクエリごとの詳細情報が表示されます。

Query Path	
And	Employee.firstname LIKE "h@%" (38374 records in 4 ms)
Or	Employee.lastname == "sm@%" (420 records in 2 ms)
	Employee.salary > 30 000 (1233902 records in 24 ms)

クエリエリアで右クリックすると、以前の有効なクエリが表示されます：

ID	firstname	lastname	salary
2	isabelle	Ma	10000
3	Narima	Tuna	15000

右側に表示されるクエリリスト：

- firstname = "h@%" AND lastname == "sm@%" OR salary > 30000
- firstname = "h@%" AND lastname == "sm@%" OR salary > 30000
- firstname = "h@%"
- firstname == "Jim"
- firstname = "M" and lastname = "@th"
- firstname = "M@%" and lastname = "@th"
- firstname = "M@%" or lastname = "@th"
- firstname = "h@%" AND lastname == "smith" OR salary > 30000

コマンドライン・インターフェース

macOS のターミナルまたは Windows のコンソールを使用して、コマンドラインによる 4D アプリケーション (4D および 4D Server) の起動ができます。この機能により、以下のことが可能になります：

- リモートからのデータベース起動。これは特に Web サーバーとして動作する 4D の管理に便利です。
- アプリケーションの自動テストの実行

基本情報

4D アプリケーションのコマンドラインは、macOS のターミナルまたは Windows のコンソールで実行できます。

- macOS では、`open` コマンドを使用します。
- Windows では、引数を直接渡すことができます。

macOS でも、パッケージ内のアプリケーションがあるフォルダー (Contents/MacOS パス) に移動することによって、引数を直接渡すことができ、エラーストリーム (stderr) にアクセスできるようになります。たとえば、4D パッケージが `MyFolder` フォルダーにある場合、次のようにコマンドラインを書く必要があります：`/MyFolder/4D.app/Contents/MacOS/4D`。しかしながら、エラーストリーム (stderr) にアクセスする必要がない場合には、`open` コマンドの使用が推奨されます。

4D アプリケーションの起動

ここでは、4D アプリケーションを起動するためのコマンドラインとサポートされている引数について説明します。

シンタックス：

```
<applicationPath> [--version] [--help] [--project] [<projectPath | packagePath | 4dlinkPath>] [--data <dataPath>] [--opening-mode interpreted | compiled] [--create-data] [--user-param <user string>] [--headless] [--datadir <datadir>] [--webadmin-settings-file] [--webadmin-access-key] [--webadmin-auto-start] [--webadmin-store-settings]
```

引数	値	説明
<code>applicationPath</code>	4D、4D Server、または組み込みアプリケーションへのパス。	アプリケーションを起動します。4D アプリケーションをダブルクリックするのと同じ動作をします。ストラクチャーファイルを指定する引数なしで呼び出された場合、アプリケーションが実行され、データベースを選択するためのダイアログボックスが表示されます。
<code>--version</code>		アプリケーションのバージョンを表示して終了します。
<code>--help</code>		ヘルプを表示して終了します。代替引数: <code>-?</code> , <code>-h</code>
<code>--project</code>	<code>projectPath</code> <code>packagePath</code> <code>4dlinkPath</code>	カレントデータファイルを開くプロジェクトファイル。ダイアログボックスは表示されません。
<code>--data</code>	<code>dataPath</code>	指定されたプロジェクトファイルで開くデータファイル。指定しない場合、4D は最後に開いたデータファイルを使用します。
<code>--opening-mode</code>	<code>interpreted</code> <code>compiled</code>	データベースをインタープリタモードまたはコンパイルモードで開くように指示します。指定のモードが利用できない場合でも、エラーは発生しません。
<code>--create-data</code>		有効なデータファイルが見つからない場合、新しいデータファイルを自動的に作成します。ダイアログボックスは表示されません。 <code>"-data"</code> 引数で渡されたファイルがあれば、4D はそれを使用します（同じ名前のファイルが既に存在する場合は上書きされます）。

引数 user-param	値 カスタムのユーザー文字列	説明 Get database parameter コマンドを通して 4D アプリケーションで利用可能な任意の文字列 (ただし文字列は予約文字である "-" から始まってはいけません)。
--headless		<p>4D、4D Server、または組み込みアプリケーションをインターフェースなし (ヘッドレスモード) で起動します。このモードでは:</p> <ul style="list-style-type: none"> ● デザインモードは使えません。データベースはアプリケーションモードで起動します。 ● ツールバー、メニューバー、MDI ウィンドウやスプラッシュスクリーンは表示されません。 ● Dock またはタスクバーにはアイコンは表示されません。 ● 開かれたデータベースは、"最近使用したデータベース" メニューに登録されません。 ● 4D 診断ファイルの記録が自動的に開始されます (SET DATABASE PARAMETER、値79 参照) ● ダイアログボックスへのコールはすべてインターフェットされ、自動的にレスポンスが返されます (例: ALERT コマンドの場合は OK、エラーダイアログの場合は Abort など)。インターフェットされたコマンド (*) は、診断ファイルに記録されます。 <p>保守上の理由から、LOG EVENT コマンドを使用して任意のテキストを標準の出力ストリームに送ることができます。ヘッドレスモードの 4D アプリケーションは、QUIT 4D を呼び出すか OS タスクマネージャーを使用することでしか終了できない点に注意が必要です。</p>
--dataless		<p>4D、4D Server、または組み込みアプリケーションをデータレスモードで起動します。データレスモードは、4D がデータを必要としないタスク (プロジェクトのコンパイルなど) を実行する場合に便利です。このモードでは:</p> <ul style="list-style-type: none"> ● コマンドラインや .4DLink ファイルで指定されていても、また CREATE DATA FILE や OPEN DATA FILE コマンドを使用していても、データを含むファイルは開かれません。 ● データを操作するコマンドはエラーを生成します。たとえば、CREATE RECORD は "このコマンドの対象となるテーブルがありません" というエラーを生成します。 <p>注記:</p> <ul style="list-style-type: none"> ● コマンドラインで引数が渡された場合、アプリケーションを終了しない限り、4D で開かれているすべてのデータベースにデータレスモードが適用されます。 ● .4DLink ファイルを使って引数が渡された場合には、データレスモードは .4DLink ファイルで指定されたデータベースにのみ適用されます。.4DLink ファイルの詳細については、プロジェクトを開く (その他の方法) を参照ください。
--webadmin-settings-file	ファイルパス	WebAdmin Web サーバー 用のカスタム WebAdmin .4DSettings ファイルのパス
--webadmin-access-key	String	WebAdmin Web サーバー 用のアクセスキー
--webadmin-auto-start	ブール	WebAdmin Web サーバー 用の自動スタートアップ設定の状態
--webadmin-store-settings		アクセスキーと自動スタートアップパラメーターを、現在使用している設定ファイル (デフォルトの WebAdmin.4DSettings ファイル、または --webadmin-settings-path パラメーターで指定されたカスタムファイル) に保存します。必要に応じて --webadmin-store-settings 引数を使用して、これらの設定を保存します。

診断ログファイル に記録することができません (ライセンス警告、変換ダイアログ、データベース選択、データファイル選択)。このような場合、エラーストリーム (stderr) とシステムのイベントログにエラーが投げられ、アプリケーションが終了します。

例題

以下の例題では、4D アプリケーションがデスクトップに保存されており、開こうとしているデータベースが "Documents" フォルダーにあるものとします。

ユーザーのカレントフォルダーは、macOS では "~" コマンドを、Windows では "%HOMEPATH%" コマンドを使用することで取得することができます。

アプリケーションを起動:

- macOS:

```
open ~/Desktop/4D.app
```

- Windows:

```
%HOMEPATH%\Desktop\4D\4D.exe
```

macOS上でパッケージファイルを指定してアプリケーションを起動:

```
yarn open ~/Desktop/4D.app --args ~/Documents/myDB.4dbase
```

プロジェクトファイルを指定してアプリケーションを起動:

- macOS:

```
yarn open ~/Desktop/4D.app --args ~/Documents/myProj/Project/myProj.4DProject
```

- Windows:

```
%HOMEPATH%\Desktop\4D\4D.exe %HOMEPATH%\Documents\myProj\Project\myProj.4DProject
```

プロジェクトファイルとデータファイルを指定してアプリケーションを起動:

- macOS:

```
open ~/Desktop/4D.app --args --project ~/Documents/myProj/Project/myProj.4DProject --data ~/Documents/da
```

- Windows:

```
%HOMEPATH%\Desktop\4D\4D.exe --project %HOMEPATH%\Documents\myProj\Project\myProj.4DProject --data %HOM  
または  
%HOMEPATH%\Desktop\4D\4D.exe /project %HOMEPATH%\Documents\myProj\Project\myProj.4DProject /data %HOMEP
```

.4DLink ファイルを指定してアプリケーションを起動:

- macOS:

```
open ~/Desktop/4D.app MyDatabase.4DLink
```

```
open "~/Desktop/4D Server.app" MyDatabase.4DLink
```

- Windows:

```
%HOMEPATH%\Desktop\4D.exe MyDatabase.4DLink
```

```
%HOMEPATH%\Desktop\4D Server.exe" MyDatabase.4DLink
```

アプリケーションをコンパイルモードで起動し、データファイルが利用できない場合には作成する:

- macOS:

```
open ~/Desktop/4D.app ~/Documents/myBase.4dbase --args --opening-mode compiled --create-data true
```

- Windows:

```
%HOMEPATH%\Desktop\4D\4D.exe %HOMEPATH%\Documents\myBase.4dbase\myDB.4db --opening-mode compiled --creat
```

プロジェクトファイルとデータファイルを指定してアプリケーションを起動し、ユーザー引数として文字列を渡す:

- macOS:

```
open ~/Desktop/4D.app --args --project ~/Documents/myProj/Project/myProj.4DProject --data ~/Documents/da
```

- Windows:

```
%HOMEPATH%\Desktop\4D\4D.exe --project %HOMEPATH%\Documents\myProj\Project\myProj.4DProject --data %HOME
```

インターフェースなしのアプリケーションを起動する (ヘッドレスモード):

- macOS:

```
open ~/Desktop/4D.app --args --project ~/Documents/myProj/Project/myProj.4DProject --data ~/Documents/da
```

```
open ~/Desktop/MyBuiltRemoteApp --headless
```

- Windows:

```
%HOMEPATH%\Desktop\4D\4D.exe --project %HOMEPATH%\Documents\myProj\Project\myProj.4DProject --data %HOME  
%HOMEPATH%\Desktop\4D\MyBuiltRemoteApp.exe --headless
```

TLSプロトコル (HTTPS)

すべての 4Dサーバーは、TLS (Transport Layer Security) プロトコルを通じて、保護モードで通信する事ができます:

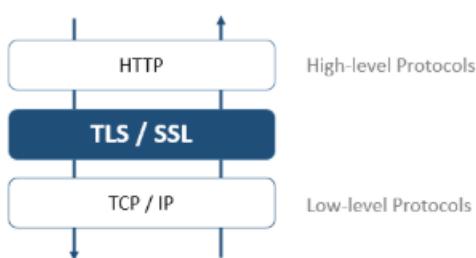
- Webサーバー
- アプリケーションサーバー (クライアントサーバー・デスクトップアプリケーション)
- SQLサーバー

概要

TLSプロトコル (SSLプロトコルの後継版) は 2つのアプリケーション、主に Webサーバーとブラウザー間でのデータ交換を保護するために設計されています。このプロトコルは幅広く使用されていて、多くの Webブラウザーとの互換性があります。

ネットワークレベルにおいては、TLSプロトコルは TCP/IPレイヤー (低レベル) とHTTP高レベルプロトコルとの間に挿入されます。TLS は主に HTTP で動作するように設計されました。

TLS を用いたネットワーク設定:



TLSプロトコルは、送信者と受信者を認証するために設計され、交換された情報の機密性と整合性を保証します:

- 認証: 送信者と受信者の ID を確認します。
- 機密性: 送信データを暗号化します。そのため第三者はメッセージを解読することができません。
- 整合性: 受信データが偶発的にまたは故意に修正されることはありません。

TLS は公開鍵暗号化技術を用います。これは、暗号化と復号化の非対称鍵のペアである公開鍵と秘密鍵に基づいています。秘密鍵はデータを暗号化するために使用されます。送信者 (Webサイト) は、それを誰にも渡しません。公開鍵は情報を復号化するために使用され、証明書を通して受信者 (Webブラウザー) へ送信されます。インターネットで TLS を使用する際、証明書は Verisign® などの認証機関を通して発行されます。Webサイトは証明書を認証機関から購入します。この証明書はサーバー認証を保証し、保護モードでのデータ交換を許可する公開鍵を格納しています。

暗号化メソッドと公開鍵および秘密鍵に関する詳細は、`ENCRYPT BLOB` コマンドの記述を参照してください。

最低バージョン

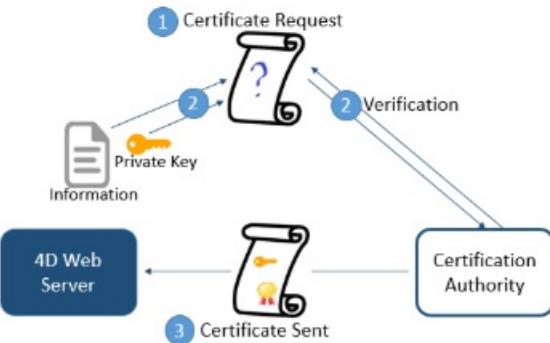
デフォルトで、4D でサポートされている最低限のバージョンは TLS 1.2 となります。この値は `SET DATABASE PARAMETER` コマンドで `Min TLS version` セレクターを使用することで変更可能です。

接続時に受け入れる [最低TLSバージョン](#) を定義することで、Webサーバーのセキュリティレベルを制御することができます。

証明書の取得方法

サーバーを保護モードで起動させるには、認証機関の電子証明書が必要です。この証明書には、サイトID や、サーバーとの通信に使用する公開鍵など、様々な情報が格納されます。そのサーバーに接続した際に、証明書がクライアント (例: Webブラウザー) へ送られます。証明書が識別され受け入れられると、保護モードで通信が開始されます。

ブラウザーは、ルート証明書がインストールされた認証機関によって発行された証明書のみを許可します。



認証機関は複数の条件によって選択されます。認証機関が一般によく知られていると、証明書は多くのブラウザによって許可されます。ただし、費用は高くなるかもしれません。

デジタル証明書の取得:

1. `GENERATE ENCRYPTION KEYPAIR` コマンドを使用して、秘密鍵を作成します。

警告: セキュリティ上の理由により、秘密鍵は常に機密でなければなりません。実際、秘密鍵は常にサーバーマシンと一緒に存在しているべきです。Webサーバーの場合、Key.pem ファイルは Project フォルダーに保存されていなければなりません。

2. 証明書のリクエストを発行するために `GENERATE CERTIFICATE REQUEST` コマンドを使用します。

3. その証明書リクエストを選択された認証機関へ送ります。

証明書リクエストを記入する際、認証機関への問い合わせが必要となる場合があります。認証機関は送信してきた情報が正確なものかを確認します。その証明書リクエストは base64 で暗号化された PKCSフォーマット (PEMフォーマット) を用いて BLOB に作成されます。この原理を使用すると、テキストとしてキーをコピー & ペーストできます。キーの内容を修正せずに認証機関に提出します。たとえば、テキストドキュメントに証明書リクエストを含んでいる BLOB を保存します (`BLOB TO DOCUMENT` コマンドを使用)。そして、コンテンツを開き、それをコピーして、認証機関へ送信するメールまたは Webフォームにペーストします。

4. 証明書を取得したら、"cert.pem" という名前でテキストファイルを作成し、その証明書の内容をそのファイルへコピーします。

証明書は様々な方法で受け取ることができます (通常は Eメールまたは HTML形式で受け取ります)。4D は証明書に関しては全プラットフォームに関連したテキストフォーマットを受け付けます (OS X、Windows、Linux、等)。ただし、証明書は PEMフォーマット、つまり base64 で PKCSエンコードされている必要があります。

CR改行コードは、それ単体ではサポートされていません。改行コードは CRLF または LF を使用してください。

5. "cert.pem" ファイルを [適切な場所](#) に保存します。

4Dサーバーが保護モードで動作するようになります。証明書は 3ヶ月から1年間の間で有効です。

インストールとアクティベーション

`key.pem` と `cert.pem` ファイルのインストール

サーバーと一緒に TLSプロトコルを使用するには、key.pem (秘密の暗号鍵を含むドキュメント) と cert.pem (証明書を含むドキュメント) が所定の場所にインストールされなければなりません。TLS を使用するサーバーによって、インストールする場所が異なります。

デフォルトの `key.pem` と `cert.pem` は 4D によって提供されています。より高レベルのセキュリティのためには、これらのファイルをご自身の証明書で置き換えることが強く推奨されます。

Webサーバーの場合

4D Webサーバーで使用するには、key.pem と cert.pem を次の場所に保存します:

- 4D (ローカル) および 4D Server では、[Project フォルダー](#) と同階層。
- 4D のリモートモードでは、これらのファイルはリモートマシンの 4D Client Database フォルダーに置かれなければなりません。このフォルダーの場所

に関する情報は、[Get 4D Folder](#) コマンドの説明を参照ください。

これらのファイルをリモートマシンに手動でコピーする必要があります。

アプリケーションサーバー（クライアントサーバー・デスクトップアプリケーション）の場合

4D アプリケーションサーバーで使用するには、key.pem と cert.pem を次の場所に保存します：

- 4D Server アプリケーションの [Resourcesフォルダー](#)。
- 各リモート4Dアプリケーションの Resources フォルダー（このフォルダーの場所に関する情報は、[Get 4D Folder](#) コマンドの説明を参照ください）。

SQLサーバーの場合

4D SQLサーバーで使用するには、key.pem と cert.pem ファイルは[Project フォルダー](#) の隣に配置する必要があります。

TLSを有効にする

key.pem と cert.pem ファイルをインストールすることにより、4Dサーバーにおける TLS の使用を可能にします。ただし、サーバーによって受け入れられるようにするには、TLS接続を有効化しなければなりません：

- 4D Webサーバーの場合、[HTTPSを有効](#) にする必要があります。HSTSオプションを設定して、HTTPモードで接続しようとするブラウザーをリダイレクトすることができます。
- アプリケーションサーバーでは、ストラクチャー設定ダイアログボックスの "C/S (クライアントサーバー)" ページで クライアント-サーバー通信の暗号化オプションを選択する必要があります。
- SQLサーバーの場合は、ストラクチャー設定ダイアログボックスの "SQL" ページで TLSを有効にする オプションを選択する必要があります。

4D Webサーバーは、セキュアな HTTPS接続のみを許可するとブラウザーに対して宣言するための HSTSオプションをサポートしています。

Perfect Forward Secrecy (PFS)

PFS は通信の中に新たなレイヤーのセキュリティを追加します。事前準備された交換鍵を使用する代わりに、PFS は Diffie-Hellman (DH) アルゴリズムを用いて通信相手同士で協同的にセッションキーを作成します。このように協同で鍵を作成することで "共有の秘密" が作成され、外部への漏洩を防ぐことができます。

サーバー上で TLS が有効化されているとき、PFS は自動的に有効されます。dhparams.pem ファイル（サーバーの DH非公開鍵を含むドキュメント）がまだ存在していない場合、4D は 2048 の鍵サイズで自動的にそれを生成します。このファイルの生成には数分間かかる可能性があります。このファイルは、[key.pem](#) および [cert.pem](#) ファイルと同じ場所に置きます。

[カスタムの暗号リスト](#) を使用していて、PFS を有効化したい場合、DH あるいは ECDH (Elliptic-curve Diffie–Hellman) アルゴリズムのエントリーがそのリストに含まれている必要があります。

4D ライセンスの管理

ディスクへのインストール終了後、4D 製品を利用するためにはアクティベーションをおこないます。[4D アカウントでサインイン](#) した場合、アクティベーションは自動的におこなわれます。

しかし、場合によってはライセンスを手動でアクティベーションする必要があります。たとえば：

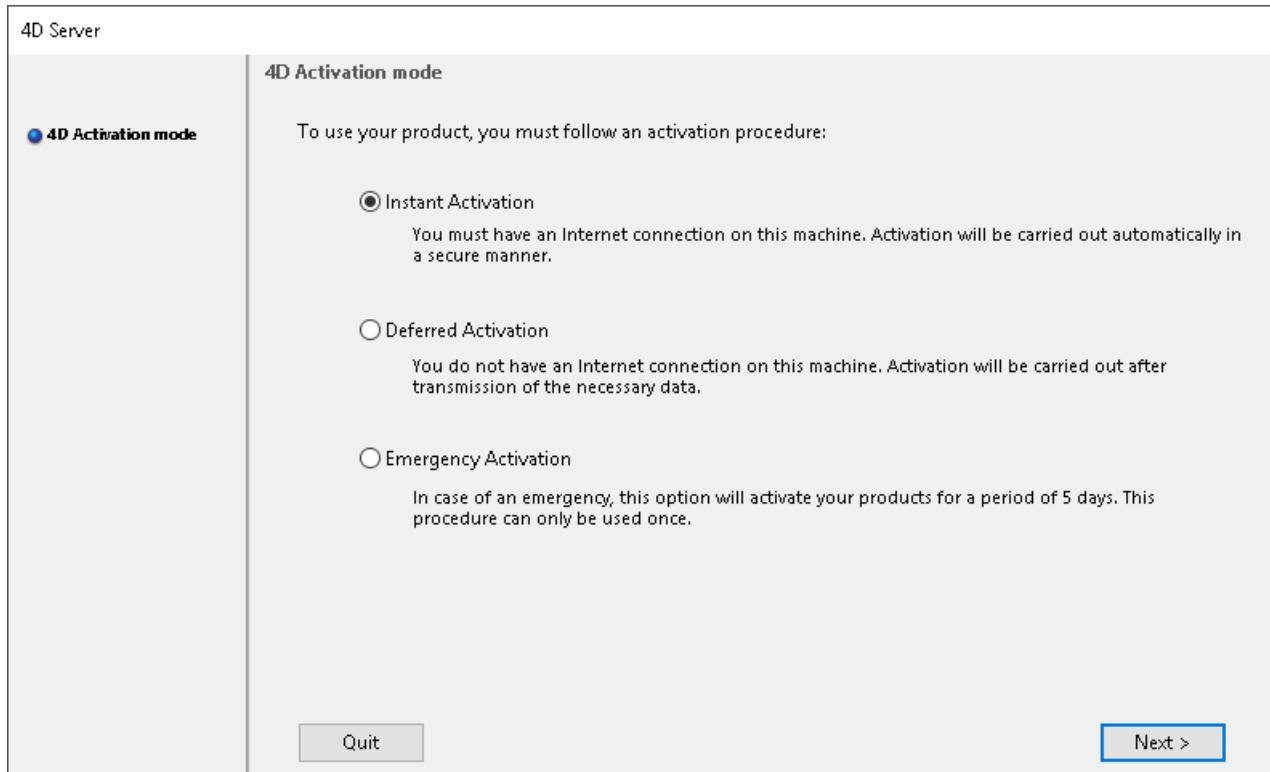
- 自動アクティベーションが可能でない場合
- 追加のライセンスを購入した場合

以下の利用モードの場合には、アクティベーションは必要はありません：

- リモートモードで利用される 4D (4D Serverへの接続)
- インタープリターモードのアプリケーションプロジェクトを開く場合で、デザインモードへはアクセスしないローカルモードの4D

初回のアクティベーション

4D でおこなう場合は、ヘルプ メニューから **ライセンスマネージャー...** を選択します。4D Server でおこなう場合は、4D Server アプリケーションを起動します。アクティベーションモードを選択するダイアログボックスが表示されます。

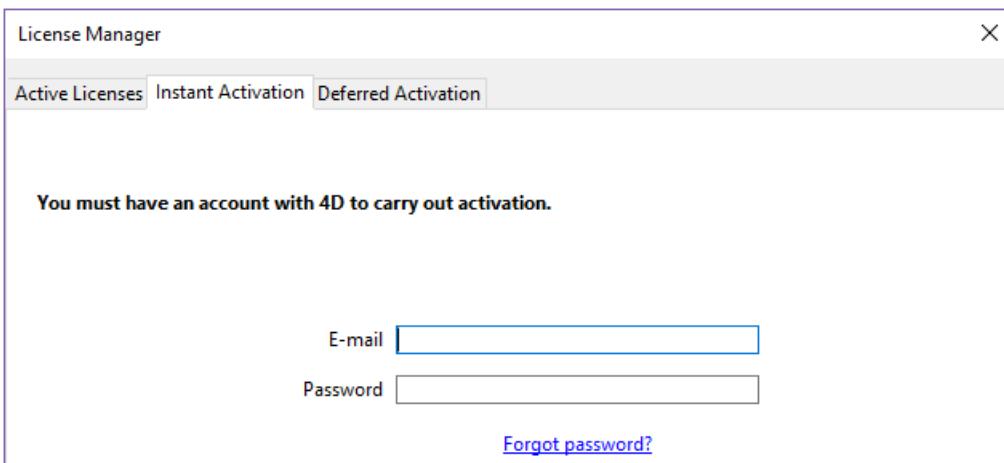


4D は 3つのアクティベーションモードを用意しています。推奨されるのは オンラインアクティベーション です。

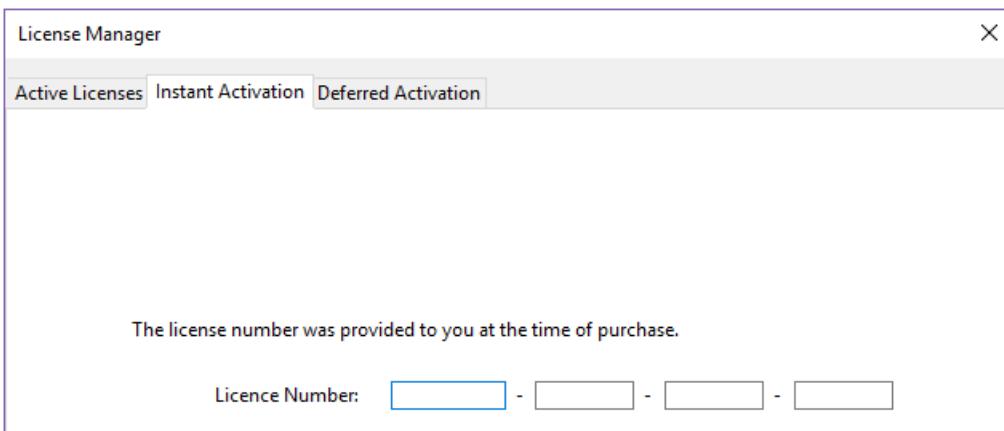
オンラインアクティベーション

ユーザーID（メールアドレスまたは 4Dアカウント）とパスワードを入力します。既存のユーザーアカウントが無い場合、まず以下のアドレスから作成する必要があります：

<https://account.4d.com/ja/login.shtml>



その後、アクティベーションする製品のプロダクト番号を入力します。このプロダクト番号は製品購入後にメールまたは郵送で提供されています。



オフラインアクティベーション

コンピューターからインターネットへのアクセスがないために [オンラインアクティベーション](#) が出来ない場合、以下の手順を踏んでオフラインアクティベーションへと進んで下さい。

1. ヘルプ メニューから "ライセンスマネージャー" を開き、オフラインアクティベーション タブを選択します。
2. ライセンス番号とメールアドレスを入力し、ファイルを生成 をクリックして IDファイル (reg.txt) を作成します。

License Manager X

Active Licenses Instant Activation Deferred Activation

Step 1 out of 3

I want to generate an ID file that I will send to 4D in order to get an activation key in return.

→ Licence Number: - - -

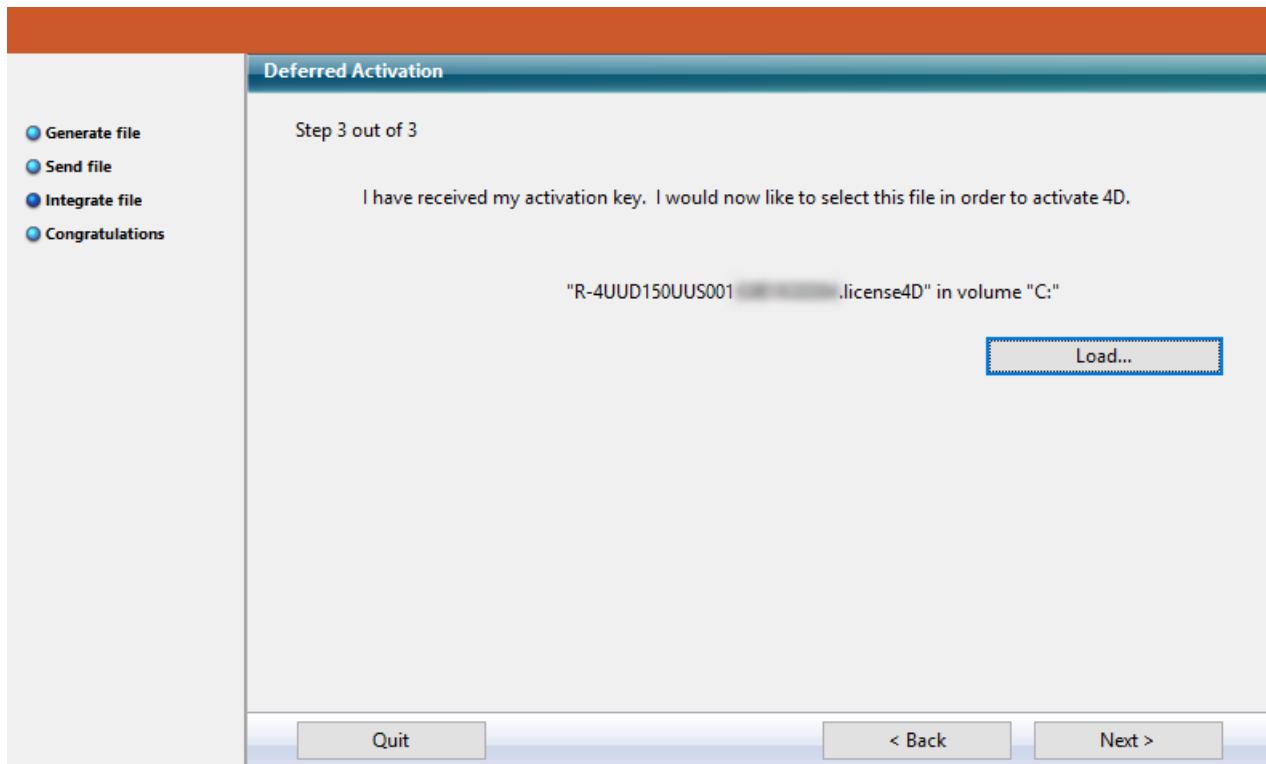
→ E-mail (mandatory):

→ Generate file...

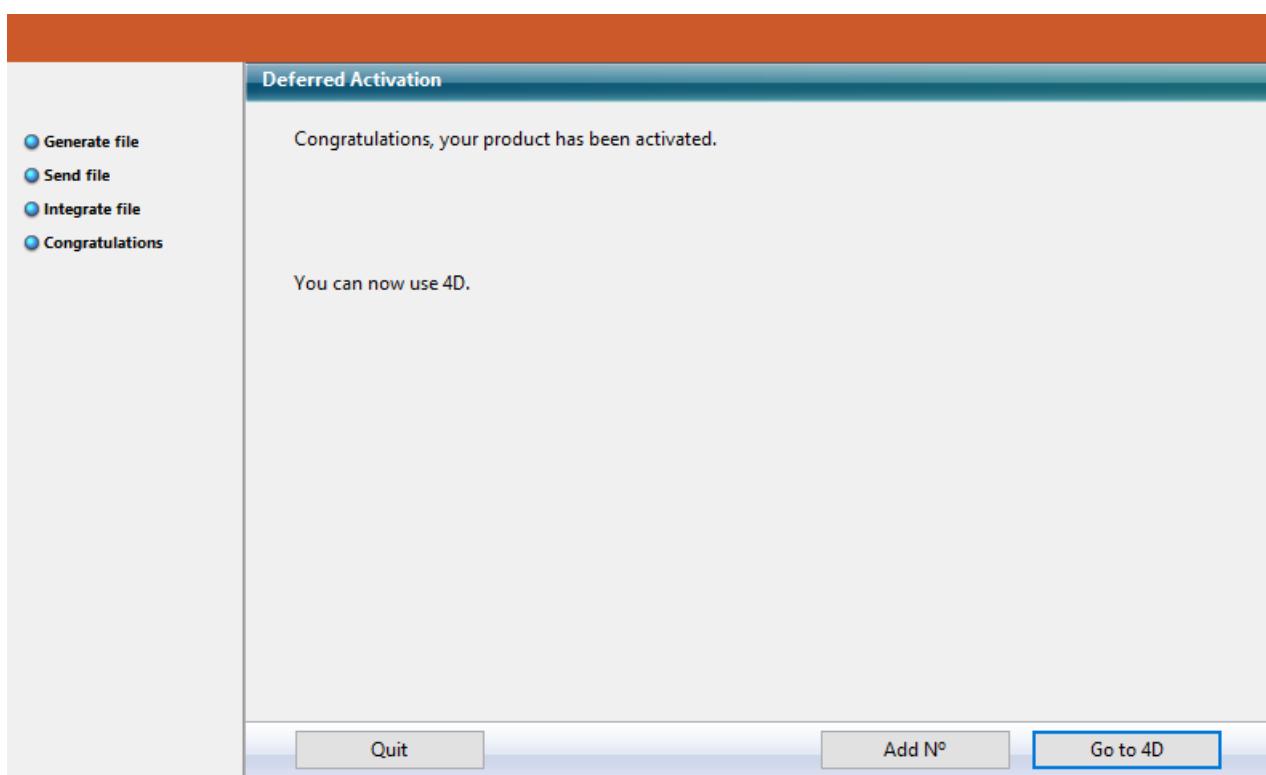
I have received my activation key. I would now like to select this key in order to activate 4D.

[< Back](#) [Next >](#) [Done](#)

3. 生成された *reg.txt* ファイルを USBドライブへと保存し、インターネット環境があるコンピューターへと移動させます。
 4. インターネット環境のあるマシンから、<https://store.4d.com/jp/activation.shtml> にログインします。
 5. Web ページ上にて、ファイルを選択... ボタンをクリックし、手順3と4で生成した *reg.txt* ファイルを選択し、Activate ボタンをクリックします。
 6. シリアルファイルをダウンロードします。
-
7. *license4d* ファイルを、何らかの共有メディアに保存し、手順1で使用している4Dマシンへと移動させます。
 8. "オフラインアクティベーション" 画面のままになっている、4D をインストールしたマシン上にて、画面上の 次へ をクリックし、次に 読み込み... ボタンをクリックして、手順7の共有メディアにある *license4d* ファイルを選択します。



ライセンスファイルが読み込まれた状態で、次へ をクリックします。



9. 他のライセンスを追加するためには 番号追加 ボタンをクリックします。これらの手順を、手順6のライセンスがすべて追加されるまで繰り返します。

これで、お使いの4Dアプリケーションのアクティベーションが完了しました。

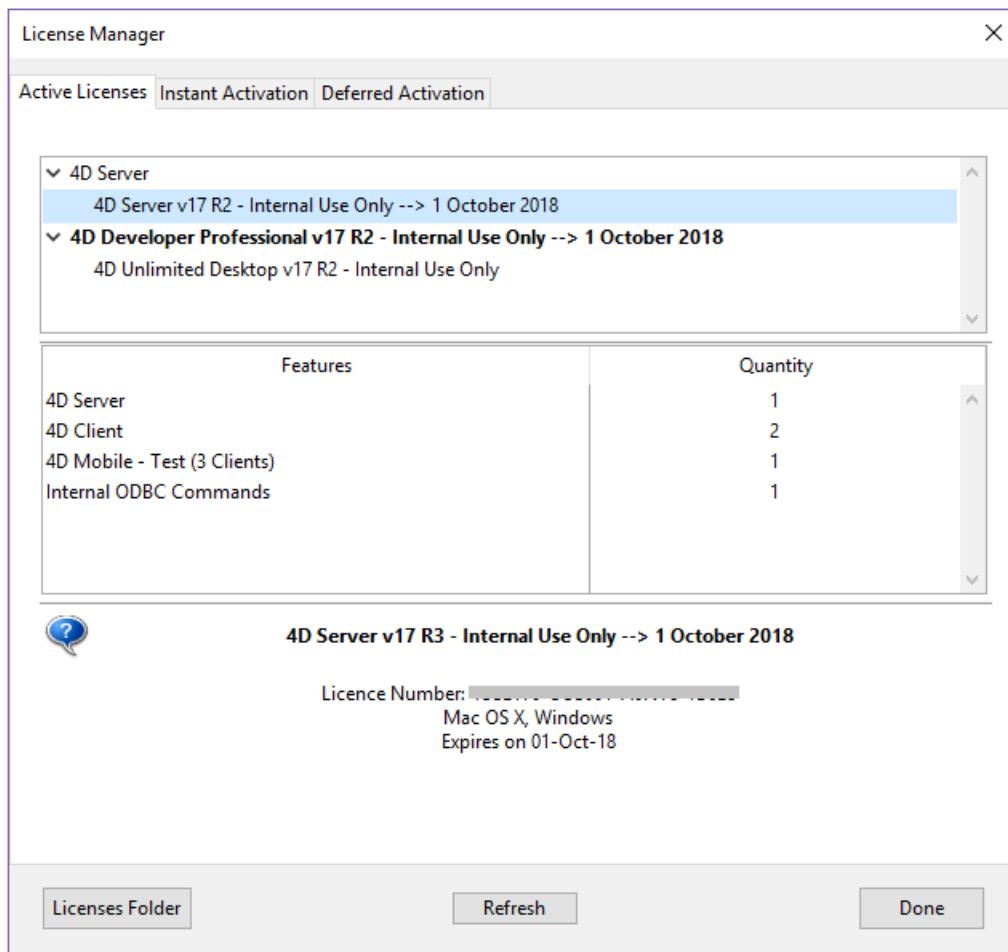
緊急アクティベーション

このモードは、特別に一時的な4Dのアクティベーションをおこなうために使用します。このアクティベーションを行うと、4Dインターネットサイトに接続せずに、最大5日間4Dを利用できます。このアクティベーションは一回のみ使用することができます。

ライセンスの追加

アプリケーションの拡張ライセンスは、いつでも追加することができます。

4D または 4D Server アプリケーションの ヘルプ メニューから ライセンスマネージャー... を選択し、更新 ボタンをクリックしてください:



このボタンを押すと 4D カスタマーデータベースに接続し、利用中のライセンスに紐付いている新しい、あるいは更新されたライセンスの自動アクティベーションがおこなわれます (利用中のライセンスは "有効なライセンス" 一覧内で 太字 で表示されているものです)。その際、4D アカウントとパスワードの入力が必要です。

- 4D Server に追加のエクスパンションを購入した場合、ライセンス番号は一切入力する必要がありません。更新 ボタンをクリックすれば、すべて完了します。
- 4D Server の初回アクティベーション時のみ、サーバーのライセンス番号を入力すれば、購入した他のエクスパンションもすべて自動的に有効化されます。

更新 ボタンは、以下のような場合に使用します:

- 追加のエクスパンションを購入したとき、またはそれをアクティベートしたいとき。
- パートナーなどの失効した有限ライセンスを更新するとき。

4D オンラインストア

4D ストアでは、4D製品の注文、アップグレード、延長、管理等をおこなうことができます。ストアは以下のアドレスからアクセス可能です:
<https://store.4d.com/jp/>

既存アカウントで ログイン するか、または 新規アカウント を作成し、画面上の指示に従ってください。

注: パスワードを忘れてしまった場合、"パスワードをお忘れの方" をクリックして下さい (ログイン画面右側のヘルプメニューにあります)。数分後に指定されたアドレスへ、パスワードリセット用の自動メールが送信されます。

ライセンス管理

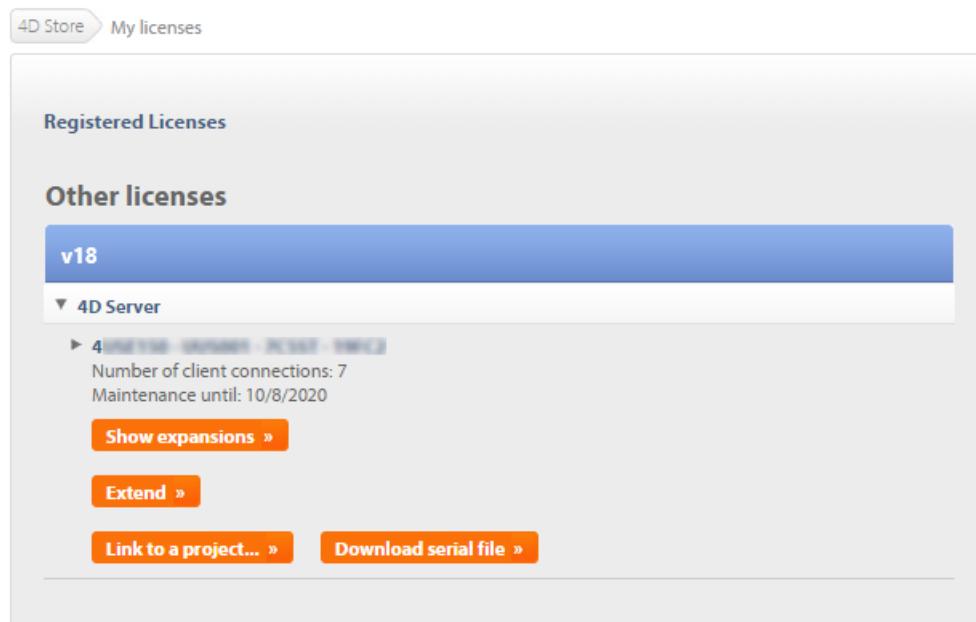
ログイン後、ページ右側のマイ・ライセンスマニュアルから ライセンスの一覧 をクリックします:

MY LICENSES

[License list »](#)
[License Registration »](#)
[Purchase an Upgrade »](#)
[Upgrade Under Maintenance »](#)

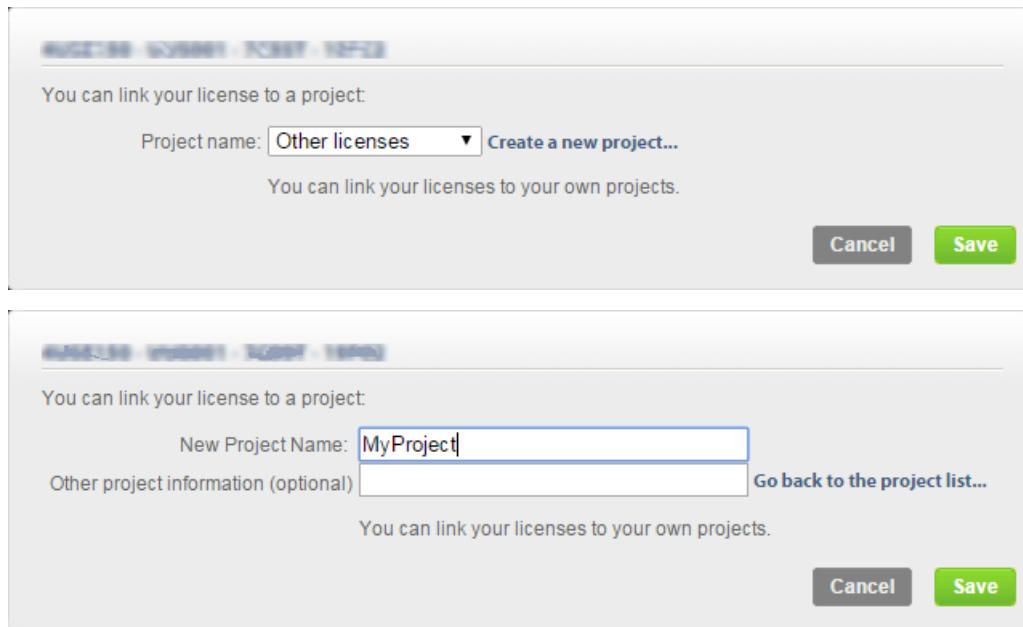
ここでは、ライセンスをプロジェクト単位でグループ化して管理することができます。

一覧から任意のライセンスを選択し、プロジェクトにリンク... > をクリックします:



The screenshot shows the 'My licenses' interface. At the top left is a '4D Store' button and a 'My licenses' link. Below is a section titled 'Registered Licenses' with a sub-section 'Other licenses'. A specific license entry for 'v18' is highlighted. This entry includes a '4D Server' section with details: 'Number of client connections: 7' and 'Maintenance until: 10/8/2020'. Below these details are three buttons: 'Show expansions »', 'Extend »', and 'Link to a project... »' (which is highlighted in orange).

既存プロジェクトを選択、または新規プロジェクトを作成します:

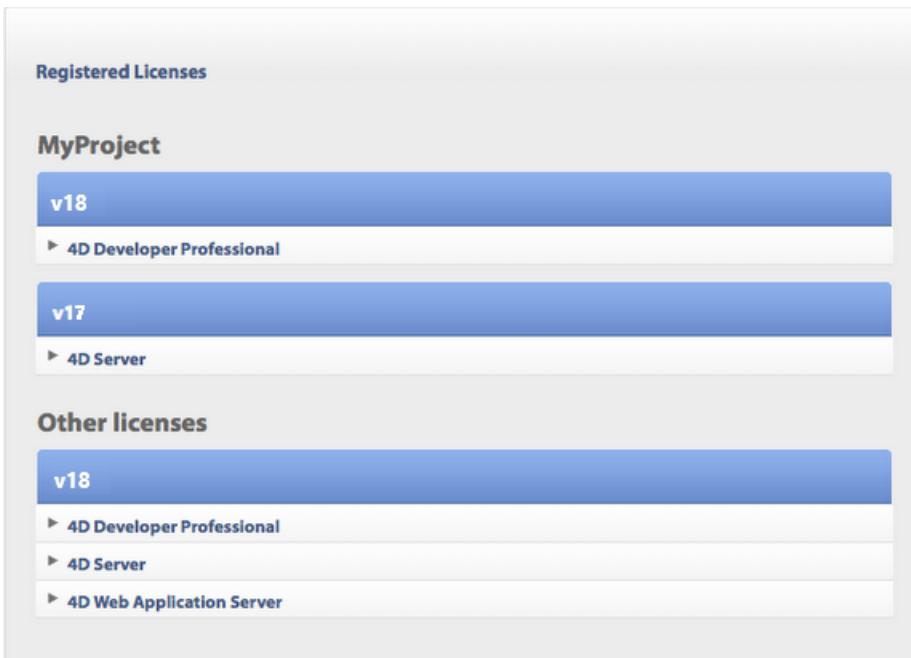


The image contains two screenshots of a 'Link to a project...' dialog.

Top Screenshot: Shows a dropdown menu with 'Other licenses' selected. Below it is a 'Create a new project...' link. A message at the bottom says 'You can link your licenses to your own projects.' with 'Cancel' and 'Save' buttons.

Bottom Screenshot: Shows a 'New Project Name:' field containing 'MyProject'. Below it is an 'Other project information (optional)' field and a 'Go back to the project list...' link. A message at the bottom says 'You can link your licenses to your own projects.' with 'Cancel' and 'Save' buttons.

プロジェクトを利用することで、必要に応じてライセンスを整理することができます:



トラブルシューティング

インストールやアクティベーションに失敗する場合は以下の表を参照してください。ほとんどの問題はこれらのケースに当てはまります:

症状	考えられる原因	解決法
4D社のサイトからインストーラーをダウンロードできません。	サイトがダウンしている、またはアンチウィルスやファイアウォールなどの影響	1- 時間を空けて再度試してください または 2- 一時的にアンチウィルスソフトやファイアウォールを無効にしてください。
ディスクに製品をインストールできません（インストールが拒否される）。	アプリケーションのインストール権限がない	アプリケーションをインストールする権限を持ったセッションを開いてください（管理者アクセス）。
オンラインアクティベーションに失敗します。	アンチウィルス、ファイアウォール、プロキシ	1- 一時的にアンチウィルスソフトやファイアウォールを無効にしてください または 2- オフラインアクティベーションを試してください。（ただし "R" バージョン用のライセンスでは利用不可）

この情報で問題が解決しない場合は、お問い合わせください。

連絡先

お買い求めいただきました製品のインストールやアクティベーションに関するご質問はフォーディー・ジャパン社、またはお住まいの地域の代理店までお寄せください。

日本にお住まいの方:

- Web: <https://jp.4d.com/technical-support>
- Tel: 03-4400-1789

4Dユーザー & グループの管理

マルチユーザー・アプリケーションにおいて、4Dはユーザーに対して標準的なアクセス権と特定の権限を与えます。ユーザー & グループシステムが起動されると、これらの標準的な権限が有効になります。

プロジェクトにおけるユーザー & グループ

プロジェクト・アプリケーション (.4DProject および .4dz ファイル) では、シングルユーザーおよびマルチユーザー環境の両方でユーザーとグループを設定することができます。ただし、アクセスシステムは 4D Server でのみ有効です。次の表は、主なユーザーとグループの機能と、それらが利用かどうかを一覧に示します：

	4D (シングルユーザー)	4D Server
ユーザーとグループの追加/編集	○	○
ユーザー/グループにサーバーアクセスを割り振る	○	○
ユーザー認証	× (すべてのユーザーがデザイナーです)	○
デザイナーへのパスワード設定によるアクセスシステムの起動	× (すべてのアクセスがデザイナーです)	○

シングルユーザー環境でのユーザー認証とアクセスコントロールについては、[シングルユーザー・アプリケーションのアクセスコントロール](#) を参照ください。

デザイナーと管理者

最も強力なユーザーは デザイナー (Designer) です。デザイナーは、アプリケーションに関するあらゆる操作をおこなうことができます。デザイナーは次のことができます：

- 制限なく、すべてのアプリケーションサーバーにアクセスする。
- ユーザーやグループを作成する。
- グループにアクセス権を割り当てる。
- デザインモードを使用する。シングルユーザー環境では、常にデザイナーアクセス権が使用されます。クライアント/サーバー環境においては、デザイナーにパスワードを割り当てることで、4Dユーザー・ログインダイアログが表示されるようになります。この環境では、デザインモードは読み取り専用です。

デザイナーの次に強力なユーザーは 管理者 (Administrator) であり、通常はパスワードアクセスシステムや管理機能を扱う役割を与えられています。

管理者は次のことができます：

- ユーザーやグループを作成する。
- 4D Server 管理ウィンドウとモニターにアクセスする。
- バックアップ、復元、サーバーの監視のため、MSC にアクセスする。

管理者は次のことができません：

- デザイナーユーザーを編集する。
- アプリケーションの保護された領域にアクセスする。とくにデザインモードが制限されている場合には、管理者はアクセスすることができません。管理者がアプリケーション内でアクセス権を得るには、1つ以上のグループに属さなければなりません。管理者はすべての新規グループに含まれますが、任意のグループから管理者の名前を取り除くことができます。

デザイナーと管理者は、すべてのアプリケーションにおいてデフォルトで利用可能です。 [ユーザー管理のダイアログボックス](#)において、デザイナーと管理者のアイコンは、それぞれ赤色と緑色で表示されます：

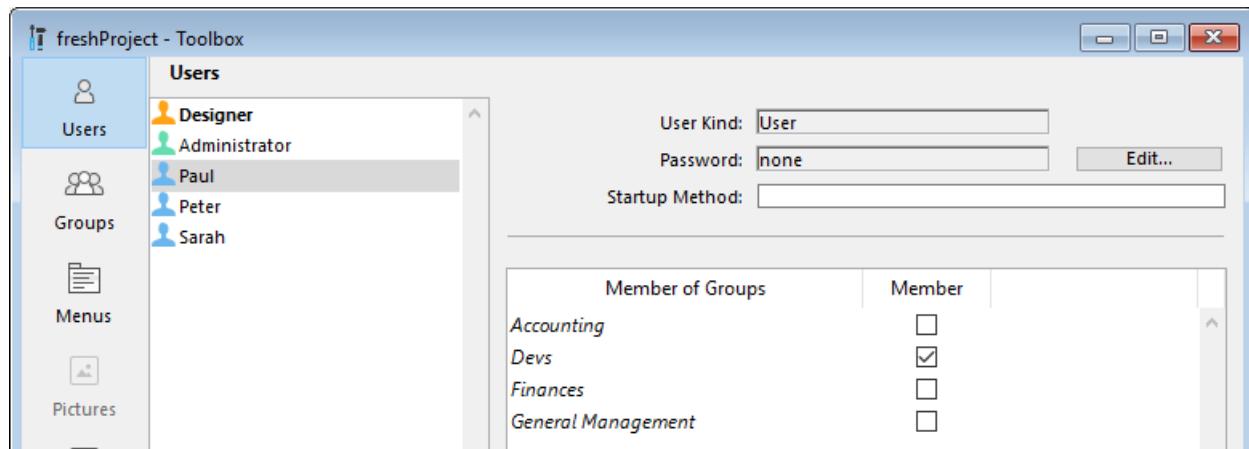
- デザイナー・アイコン: 
- 管理者・アイコン: 

デザイナーと管理者の名前は変更することができます。ランゲージにおいて、デザイナーと管理者の ID 値は、常に 1 と 2 に設定されます。

デザイナーと管理者は、それぞれ 16,000 のグループと 16,000 のユーザーを作成することができます。

ユーザー エディター

ユーザーのエディターは 4Dのツールボックスにあります。



ランタイムにおいてユーザーとグループのエディターを表示させるには **EDIT ACCESS** コマンドを使用します。ユーザーとグループの設定は、アプリケーション実行中でも **Users and Groups** テーマの 4Dランゲージコマンドを使って編集することができます。

ユーザーの追加と変更

ユーザー エディターを使用して、ユーザー アカウントの作成やプロパティの設定、各グループへの割り当てをおこないます。

ユーザーを追加するには：

1. デザイン メニューから ツールボックス > ユーザー を選択、または 4Dツールバーの ツールボックス ボタンをクリックします。4Dはユーザー エディターを表示します。
2. ユーザーリストには、**デザイナーと管理者** を含むすべてのユーザーが表示されます：
3. ユーザーリストの下にある追加ボタン **+** をクリックします。または
ユーザーリスト上で右クリックし、コンテキストメニューから **追加** または **複製** を選択する。

複製 コマンドを使用すると、同じ特性を持つ複数のユーザーを素早く作成することができます。

4D は新規ユーザーをリストに追加し、デフォルトとして "新規ユーザー-X" という名前を設定します。

3. 新しいユーザー名を入力します。この名前は、ユーザーがアプリケーションを開く際に使用されます。ユーザー名をいつでも変更することができます。変更するにはコンテキストメニューの **名称変更** コマンドを使用するか、Alt+クリック (Windows) または Option+クリック (macOS) ショートカットを使用、または変更したい名前を 2回クリックします。
4. ユーザーのパスワードを設定するには、プロパティエリアで **編集...** ボタンをクリックして、ダイアログボックスの 2つのパスワード欄に同じパスワードをそれぞれ入力します。パスワードには 15桁までの英数字を使用することができます。パスワードでは文字の大小が区別されます。

ストラクチャ設定の "セキュリティ" ページで許可されていれば、ユーザーは自分のパスワードを変更できます。また、パスワードは **CHANGE PASSWORD** コマンドを使って変更することもできます。

5. グループメンバー表を用いて、そのユーザーが所属するグループを設定します。メンバーカラムの該当するオプションをチェックして、選択したユーザーをグループに対して追加・削除することができます。

[グループページ](#) を使用して、各グループの所属ユーザーを設定することもできます。

ユーザーの削除

ユーザーを削除するには、そのユーザーを選択してから削除ボタンをクリックするか、またはコンテキストメニューの **削除** コマンドを使用します。

削除されたユーザー名は、その後ユーザー エディターには表示されません。削除されたユーザーの ID番号は、新規アカウント作成の際に再度割り当てられるという点に注意してください。

ユーザープロパティ

- ユーザーの種類: "デザイナー"、"管理者"、または (それ以外のすべてのユーザーの場合にあ) "ユーザー"
- 開始メソッド: ユーザーがアプリケーションを開いたときに自動実行されるメソッドの名称 (任意) このメソッドを使って、たとえばユーザー設定をロードできます。

グループエディター

グループのエディターは 4Dのツールボックスにあります。

グループの設定

グループエディターを使用して、各グループ内に納める要素 (ユーザーや他のグループ) を設定したり、プラグインへのアクセス権を割り当てることができます。

グループは一旦作成されると、削除できないということに留意が必要です。グループを使用したくない場合は、そのグループの所属ユーザーをすべて取り除きます。

グループを作成するには:

- デザインメニューから ツールボックス>ユーザーグループを選択、または 4Dツールバーの ツールボックス ボタンをクリックし、グループページを開きます。4D はグループエディターウィンドウを表示します: グループリストには、アプリケーションプロジェクトのすべてのグループが表示されます。
- グループリストの下にある追加ボタン



をクリックします。

または

グループリスト上で右クリックし、コンテキストメニューから 追加 または 複製 を選択します。

複製コマンドを使用すると、同じ特性を持つ複数のグループを素早く作成することができます。

4D は新規グループをリストに追加し、デフォルトとして "新規グループX" という名前を設定します。

- 新しいグループの名前を入力します。グループ名には 15桁までの文字を使用できます。グループ名をいつでも変更することができます。変更するにはコンテキストメニューの 名称変更 コマンドを使用するか、Alt+クリック (Windows) または Option+クリック (macOS) ショートカットを使用、または変更したい名前を 2回クリックします。

ユーザーやグループをグループに入れる

任意のユーザーやグループをグループ内に配置することができます。さらに、そのグループ自体を他のいくつかのグループ内に入れることも可能です。必ずしもユーザーをグループに入れる必要はありません。

ユーザーやグループをグループに配置するには、当該グループのユーザー/グループ一覧にてメンバー欄にチェックを入れます:

	User / Group	Member
Administrator		<input checked="" type="checkbox"/>
Designer		<input type="checkbox"/>
New user		<input type="checkbox"/>
Paul		<input type="checkbox"/>
Peter		<input checked="" type="checkbox"/>
Sarah		<input type="checkbox"/>
Finances		<input checked="" type="checkbox"/>
General Management		<input checked="" type="checkbox"/>
Devs		<input type="checkbox"/>
Admins		<input type="checkbox"/>

ユーザー名をチェックすると、そのユーザーがグループに追加されます。グループ名をチェックした場合は、そのグループの全ユーザーがグループへ追加されます。メンバーの一員となったユーザーやグループには、そのグループに割り当てられたものと同じアクセス権が与えられます。

グループを別のグループ内に入れることにより、ユーザーの階層構造が作成されます。別のグループの配下に入れられたグループのユーザーは、両グループのアクセス権を保持します。後述の [アクセス権の階層構造](#) を参照してください。

ユーザー や グループ を グループ から 取り除くには、ユーザー / グループ 一覧で チェック を 解除します。

プラグインやサーバーにグループを割り当てる

プロジェクトにインストールされたプラグインへのアクセス権をグループに割り当てることができます。これには 4D のプラグインと任意のサードパーティーブラグインが含まれます。

プラグインへのアクセス権を割り当てるとき、所有するプラグインライセンスの使用を管理できるようになります。プラグインのアクセスグループに属さないユーザーは、そのプラグインをロードすることができません。

使用されたライセンスは 4Dセッションの間、当該グループに所属する 4Dユーザー アカウントに紐づけられます。

ツールボックスのグループページにある "プラグイン" エリアには、4Dアプリケーションによりロードされたプラグインがすべて表示されます。プラグインへのアクセス権をグループに与えるには、該当するオプションをチェックします。

Plug-in	Access
4D Client Web Server	<input type="checkbox"/>
4D Client SOAP Server	<input type="checkbox"/>
4D Write PRO	<input checked="" type="checkbox"/>
4D View PRO	<input type="checkbox"/>

4D Client Web Server や 4D Client SOAP Server 項目を使用し、リモートモードの 4D がそれぞれ Web および SOAP (Webサービス) 公開をおこなえるかどうかを管理することができます。これらのライセンスは 4D Server 側ではプラグインライセンスとしてみなされます。したがって、プラグインと同じ方法で、これらのライセンスの使用権を特定のユーザーグループに限定することができます。

アクセス権の階層構造

アプリケーションのセキュリティを確保し、ユーザーに異なるアクセスレベルを提供する最も効果的な方法は、アクセス権の階層構造を利用することです。ユーザーを適切なグループに割り振り、各グループをネストすることで、アクセス権の階層構造を形成できます。この節では、このような構造の取り扱いについて説明します。

この例題では、ユーザーは担当業務に応じて 3つあるグループの 1つに割り振られます。データ入力担当のユーザーは、Accounting (会計) グループに割り当てます。レコードの更新や無効データの削除などデータ管理を担当するユーザーは、Finances (財務) グループに割り当てます。検索の実行や分析レポートの印刷などデータ分析を担当するユーザーは、General Management (総合管理) グループに割り当てます。

割り当て完了後は、各グループのユーザーに権限が正しく配分されるようにグループをネストします。

- General Management グループには "高レベル" のユーザーだけが含まれます。

Groups

- Accounting
- Admins
- Devs
- Finances
- General Management

User / Group	Member
Administrator	<input checked="" type="checkbox"/>
Designer	<input type="checkbox"/>
Paul	<input type="checkbox"/>
Peter	<input type="checkbox"/>
Sarah	<input type="checkbox"/>
Accounting	<input type="checkbox"/>
Finances	<input type="checkbox"/>
Devs	<input type="checkbox"/>
Admins	<input type="checkbox"/>

Plug-in	Access
4D Client Web Server	<input type="checkbox"/>
4D Client SOAP Server	<input type="checkbox"/>
4D Write PRO	<input type="checkbox"/>
4D View PRO	<input type="checkbox"/>

- Financesグループには、データ管理ユーザーと General Managementグループが含まれます。したがって、General Managementグループのユーザーは Financesグループの権限も保持します。

Groups

- Accounting
- Admins
- Devs
- Finances
- General Management

User / Group	Member
Administrator	<input checked="" type="checkbox"/>
Designer	<input type="checkbox"/>
Paul	<input type="checkbox"/>
Peter	<input type="checkbox"/>
Sarah	<input type="checkbox"/>
Accounting	<input type="checkbox"/>
General Management	<input checked="" type="checkbox"/>
Devs	<input type="checkbox"/>
Admins	<input type="checkbox"/>

Plug-in	Access
4D Client Web Server	<input type="checkbox"/>
4D Client SOAP Server	<input type="checkbox"/>
4D Write PRO	<input type="checkbox"/>
4D View PRO	<input type="checkbox"/>

- Accountingグループには、データ入力を起こすユーザーと Financesグループが含まれます。したがって、Financesグループのユーザーと General Managementグループのユーザーは Accountingグループの権限も利用できます。

The screenshot shows the 'Groups' tab in the 4D Toolbox. On the left, a sidebar lists various categories: Users, Groups (selected), Menus, Pictures, Help Tips, Lists, Style Sheets, Filters, and Resources. Below the sidebar are buttons for adding (+), removing (-), and filtering (magnifying glass).

Groups section:

User / Group	Member
Administrator	<input checked="" type="checkbox"/>
Designer	<input type="checkbox"/>
Paul	<input checked="" type="checkbox"/>
Peter	<input type="checkbox"/>
Sarah	<input type="checkbox"/>
Finances	<input checked="" type="checkbox"/>
General Management	<input checked="" type="checkbox"/>
Devs	<input type="checkbox"/>
Admins	<input type="checkbox"/>

Plug-in section:

Plug-in	Access
4D Client Web Server	<input type="checkbox"/>
4D Client SOAP Server	<input type="checkbox"/>
4D Write PRO	<input checked="" type="checkbox"/>
4D View PRO	<input type="checkbox"/>

所属ユーザーの責務に基づいて、各グループに割り当てるアクセス権を決定します。

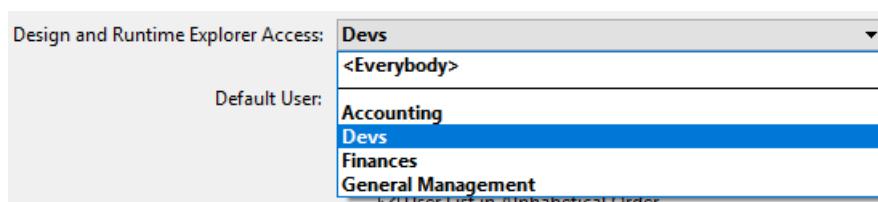
このような階層システムを使用すると、新規ユーザーに割り当てるべきグループがわかりやすくなります。各ユーザーを 1つのグループに割り当てるだけで、グループの階層を介してアクセス権を決定できます。

権限を割り当てる

グループには、アプリケーションの特定機能へのアクセス権が割り当てられます:

- デザイン環境やランタイムエクスプローラー
- HTTPサーバー
- RESTサーバー
- SQLサーバー

これらのアクセス権はストラクチャー設定で定義します。次の図は、デザインおよびランタイムエクスプローラーアクセス権を "Devs" グループに割り当てる様子を表しています (データベース設定の "セキュリティ" タブ):



また、グループを使って [利用可能なライセンスを割り当てる](#) こともできます。この割り当ては、グループエディターで定義します。

Directory.json ファイル

ユーザー、グループ、およびそれらのアクセス権は、directory.json という名称の専用のプロジェクトファイルに保存されます。

必要に応じて、このフォルダーは次の場所に保存することができます:

- すべてのデータファイルについて同じディレクトリを使用する場合 (または 1つのデータファイルだけを使用する場合)、ユーザー設定フォルダー ("Project" フォルダーと同じ階層 の "Settings" フォルダー) に directory.json ファイルを保存します (デフォルトの場所)。

- データファイルごとに特定のディレクトリファイルを使用する場合は、directory.json ファイルをデータ設定フォルダー、つまり "Data" フォルダーの "Settings" フォルダー に格納します。directory.json ファイルがこの場所に保存されている場合、ユーザー設定フォルダーのファイルよりも優先されます。アプリケーションをアップグレードしても、このカスタム/ローカルなユーザー & グループ設定はそのままです。

パスワードやグループメンバーシップを運用環境において安全に変更できるようにするには、ビルト時に [対応するビルトアプリケーションオプション](#) を使用して、サーバーアプリケーションに directory.json ファイルを含めることができます。

情報ページ

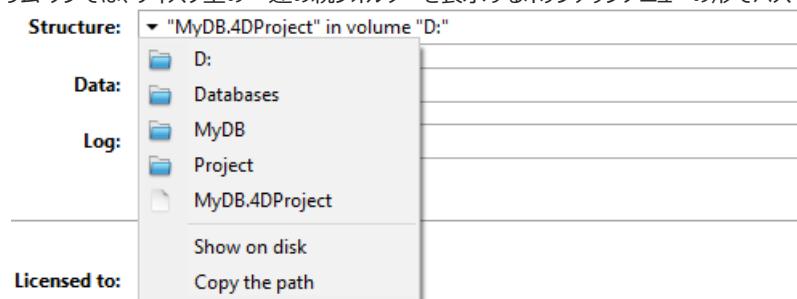
情報ページは 4D環境、システム環境、データベースおよびアプリケーションファイルについての情報を提供します。各ページは、ウィンドウ上部にあるタブコントロールを使って切り替えできます。

プログラム

このページにはアプリケーションならびにアクティブな 4Dフォルダーの名前、バージョンおよび場所を表示します（アクティブ4Dフォルダーについては [4Dランゲージリファレンス の Get 4D folder コマンドを参照ください](#)）。

ウィンドウの中央部は、プロジェクトならびにデータファイルとログファイル（あれば）の名前および場所を表示します。ウィンドウの下部は、4Dライセンスフォルダーの名前、ライセンスのタイプ、および、カレント 4Dユーザーの名前を表示します。

- パス名の表示と選択：プログラム タブでは、ディスク上の一連の親フォルダーを表示するポップアップメニューの形でパス名が示されます：



メニュー項目（ディスクまたはフォルダー）を選択した場合、そのパスが新しいシステムウィンドウで開かれます。パスをコピー コマンドは、システムのディレクトリ区切り文字を使用して、完全なパス名をクリップボードにテキストとしてコピーします。

- ライセンスフォルダー：ライセンスフォルダー ボタンをクリックすると、新しいシステムウィンドウにアクティブなライセンスフォルダーの中身を表示します。インストールされた 4D環境用のライセンスファイルはすべてこのフォルダーに格納されていなければなりません。ファイルを Web ブラウザーで開くと、ライセンスの情報が表示されます。ライセンスフォルダーの場所はバージョンや OS により異なります。このフォルダーの場所については [Get 4D folder コマンドの説明を参照してください](#)。注：上部メニューの "ヘルプ > ライセンスマネージャー..." からアクセスできるダイアログボックスにも同じボタンがあります。

テーブル

このページでは、データベース内のテーブルの概要を示します：

Maintenance and Security Center

Information

Program Tables Data

ID	Tables	Records	Fields	Indexes	Encryptable	Encrypted	Address Table Size
1	Employee	5 083	3	1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	5 083
2	Company	2 520	2	1	<input type="checkbox"/>	<input type="checkbox"/>	2 520
3	Cities	2 575	2	0	<input type="checkbox"/>	<input type="checkbox"/>	2 575
Total		10 178	7	2	1	1	10 178

このページの情報は、標準モードおよびメンテナンスマードの両方で利用可能です。

このページにはデータベースのすべてのテーブル（非表示のテーブルも含む）とそれらの特徴が表示されます：

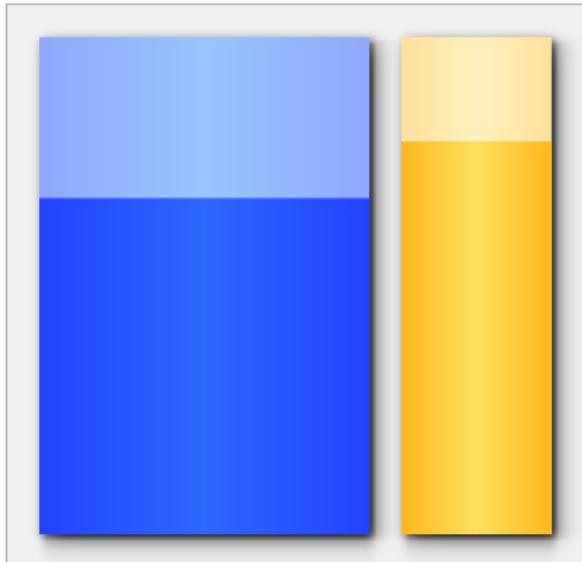
- ID: テーブルの内部番号
- テーブル: テーブル名。削除されたテーブルの名前は括弧付きで表示されます（ゴミ箱の中に残っている場合）。
- レコード: テーブル内の総レコード数。レコードが破損していたり読み込めなかった場合には、数字の代わりに *Error* が表示されます。この場合、検証と修復ツールの使用を検討してください。
- フィールド: テーブル内のフィールド数。非表示のフィールドはカウントされますが、削除されたフィールドはカウントされません。
- インデックス: テーブル内のあらゆるインデックスの数
- 暗号化可能: チェックされていれば、ストラクチャーレベルにおいてこのテーブルは 暗号化可能 属性が選択されています（デザインリファレンスマニュアルの [暗号化可能](#) の項目を参照ください）。
- 暗号化済み: チェックされていれば、テーブルのレコードはデータファイルにおいて暗号化されています。 注： 暗号化可能と暗号化済みオプション間において整合性が取れていない場合、必ず MSC の [暗号化](#) ページにてデータファイルの暗号化状態を確認してください。
- アдресステーブルサイズ: 各テーブルのアドレステーブルのサイズ。アドレステーブルとは、テーブル内で作成される各レコードにつき 1つの要素を保存する内部テーブルのことです。これはレコードとその物理アドレスをつなげる働きをします。パフォーマンス上の理由から、レコードが削除されてもリサイズはされず、そのためそのサイズはテーブル内のカレントレコード数とは異なる場合があります。この差異が著しく大きい場合、"アドレステーブルを圧縮" オプションをチェックした状態でデータ圧縮を実行することで、アドレステーブルサイズを最適化することができます（[圧縮](#) ページを参照してください）。注：アドレステーブルサイズとレコード数の差異は、キャッシュフラッシュの途中での事象によるものである可能性もあります。

データ

データ ページには、データファイルの空き/使用済み容量の情報が表示されます。

このページには、メンテナンスマードではアクセスできません。

この情報はグラフ形式で提供されます：



The overall size of these files (data+index) is 576,93 MB.
The global percentage used is 71%.

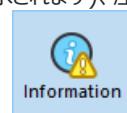


It is recommended that you compact this data file.

This will reduce the file size on disk and may noticeably increase the execution speed of the application.

このページに表示される情報には、データファイル外に格納されたデータは反映されません（[データをデータファイル外に保存](#) 参照）。

断片化があまりにも進んだファイルはディスク、そしてデータベースのパフォーマンスを低下させます。使用率が低すぎる場合、4Dは警告アイコンを表示して（このアイコンは情報ページボタンと対応するファイルタイプのタブに表示されます）、圧縮が必要であることを警告します：



警告アイコンは [圧縮](#) ページボタンにも表示されます：



ログ解析ページ

ログ解析ページを使用して、カレントログファイルに記録された内容を見ることができます。この機能はアプリケーション利用状況の解析、エラーや不具合の原因となった処理を探すなどの場合に役立ちます。クライアント/サーバーモードの場合、各クライアントマシンごとの操作を検証することもできます。

データベースのデータに対しておこなわれた操作をロールバックさせることもできます。詳細は [ロールバック](#) ページを参照してください。

Maintenance and Security Center

Activity analysis

The list below shows all the performed operations recorded in the log file since the last backup.

Operation ID	Action Sequence	Table	Primary...	Process ID	Size	Date	Hour	System User	4D User	Values	Record
792	Addition	Parts	198	26	50	16/12/2019	16:37	aschmitt	Main user	198 ;	197
793	Sequence	Invoices		26		16/12/2019	16:37	aschmitt	Main user	200	
794	Addition	Invoices	199	26	54	16/12/2019	16:37	aschmitt	Main user	199 ; ; ;	198
795	Sequence	Parts		26		16/12/2019	16:37	aschmitt	Main user	200	
796	Addition	Parts	199	26	52	16/12/2019	16:37	aschmitt	Main user	199 ;	198
797	Sequence	Invoices		26		16/12/2019	16:37	aschmitt	Main user	201	
798	Addition	Invoices	200	26	54	16/12/2019	16:37	aschmitt	Main user	200 ; ; ;	199
799	Sequence	Parts		26		16/12/2019	16:37	aschmitt	Main user	201	
800	Addition	Parts	200	26	50	16/12/2019	16:37	aschmitt	Main user	200 ;	199
801	Sequence	Invoices		26		16/12/2019	16:37	aschmitt	Main user	202	
802	Addition	Invoices	201	26	52	16/12/2019	16:37	aschmitt	Main user	201 ; ; ;	200
803	Sequence	Parts		26		16/12/2019	16:37	aschmitt	Main user	202	
804	Addition	Parts	201	26	48	16/12/2019	16:37	aschmitt	Main user	201 ;	200
805	Sequence	Invoices		26		16/12/2019	16:37	aschmitt	Main user	203	
806	Addition	Invoices	202	26	54	16/12/2019	16:37	aschmitt	Main user	202 ; ; ;	201
807	Sequence	Parts		26		16/12/2019	16:37	aschmitt	Main user	203	
808	Addition	Parts	202	26	50	16/12/2019	16:37	aschmitt	Main user	202 ;	201
809	Sequence	Invoices		26		16/12/2019	16:37	aschmitt	Main user	204	
810	Addition	Invoices	203	26	52	16/12/2019	16:37	aschmitt	Main user	203 ; ; ;	202
811	Sequence	Parts		26		16/12/2019	16:37	aschmitt	Main user	204	
812	Addition	Parts	203	26	48	16/12/2019	16:37	aschmitt	Main user	203 ;	202

Analyze Browse Export...

ログファイルに記録された操作は行として表示されます。各操作の様々な情報が列に表示されます。ヘッダーをドラッグすることによって列の並び順を変えることができます。

この情報を使用して各操作のソースとコンテキストを識別できます:

- 操作: ログファイル中での一連の操作番号
- アクション: データに対しておこなわれた操作のタイプ。この列には以下の操作のいずれかが記録されます:
 - データファイルを開く: データファイルを開いた
 - データファイルを閉じる: 開いたデータファイルを閉じた
 - コンテキストの作成する: 実行コンテキストを指定するプロセスを作成した
 - コンテキストを閉じる: プロセスを閉じた
 - 追加: レコードを作成、格納した
 - BLOB を追加: BLOBフィールドに BLOB を格納した
 - 削除: レコードを削除した
 - 更新: レコードを更新した
 - トランザクションの開始: トランザクションを開始した
 - トランザクションの受け入れ: トランザクションを受け入れた

- トランザクションのキャンセル: トランザクションをキャンセルした
- コンテキストの更新: 追加データを変更した (例: `CHANGE CURRENT USER` あるいは `SET USER ALIAS` の呼び出し)
- テーブル: 追加/削除/更新されたレコードまたは BLOB の所属テーブル
- プライマリーキー/BLOB: 各レコードのプライマリーキーのコンテンツ (プライマリーキーが複数のフィールドから構成されているときには、値はセミコロンで区切られています)、またはオペレーションに関連した BLOB のシーケンス番号
- プロセス: 処理が実行された内部プロセス番号。この内部番号は処理のコンテキストに対応します。
- サイズ: 操作により処理されたデータのサイズ (バイト単位)
- 日付と時刻: 処理が実行された日付と時刻
- システムユーザー: 操作を実行したユーザーのシステム名。クライアント/サーバーモードでは、クライアントマシン名が表示されます。シングルユーザー モードでは、ユーザーのセッション名が表示されます。
- 4Dユーザー: 操作を実行したユーザーの 4Dユーザー名。ユーザーに対してエイリアスが設定されていた場合、4Dユーザー名の代わりのそのエイリアスが表示されます。
- 値: レコードの追加や更新の場合、フィールドの値。値はセミコロン ";" で区切られます。文字形式に表現できる値のみを表示します。
注: データベースが暗号化されており、開かれたログファイルに対応する有効なデータキーが提供されていない場合、暗号化された値はこのカラムには表示されません。
- レコード: レコード番号

選択したアプリケーションのカレントログファイル (デフォルトで "データファイル名.journal" というファイル名) の内容を更新するには 解析 をクリックします。ブラウズボタンをクリックすると、アプリケーションの他のログファイルを選択できます。書き出し... ボタンを使用してファイルの内容をテキストとして書き出せます。

検査ページ

このページでは、データおよび構造上の整合性を検査できます。検査は、レコードおよびインデックスについて実行できます。この機能は検査のみをおこないます。エラーが見つかり修復が必要な場合は [修復ページ](#) を使用するよう表示されます。

アクション

このページには、検査機能に直接アクセスするための、次のアクションボタンが置かれています。

データベースが暗号化されている場合、検査の中には暗号化されたデータの整合性の評価も含まれます。有効なデータキーがまだ提供されていない場合、パスフレーズ、あるいはデータキーを要求するダイアログが表示されます。

- レコードとインデックスを検査: 全体のデータ検査処理を開始します。
- レコードのみを検査: レコードのみの検査処理を開始します (インデックスは検査されません)。
- インデックスのみを検査: インデックスのみの検査処理を開始します (レコードは検査されません)。

レコードとインデックスの検査は、テーブルごとに検査する詳細モードでおこなうこともできます(後述の "詳細" の章を参照してください)。

ログファイルを開く

要求された検査に関係なく、4D はアプリケーションの `Logs` フォルダーにログファイルを生成します。このファイルには実行された検査の内容が記録され、エラーがあればそれも示されます。問題がない場合は [OK] が表示されます。このファイルは XML形式で、ファイル名は `ApplicationNameVerify_Logyyyy-mm-dd hh-mm-ss.xml` となり、それぞれ以下の要素が入ります:

- `ApplicationName` は拡張子を除いたプロジェクトファイルの名前です (例: "Invoices" 等)
- `yyyy-mm-dd hh-mm-ss` はファイルのタイムスタンプです。これはローカルのシステム時間でメンテナスオペレーションが開始された時刻に基づいています (例: "2019-02-11 15-20-45")。

ログファイルを開く ボタンをクリックすると、4Dはマシンのデフォルトブラウザーを使用して直近のログファイルを開きます。

詳細

テーブルリスト ボタンは、検査するレコードおよびインデックスを選択するために使用する詳細ページを表示します:

検査する項目を指定することにより、検査処理にかかる時間を短縮できます。

リストには、データベースの全テーブルが表示されます。各テーブルに対して、検査対象をレコードやインデックスに限定できます。三角形のアイコンをクリックしてテーブルまたはインデックス付フィールドの内容を展開し、要件に応じてチェックボックスにチェックを入れたり解除したりします。デフォルトでは、すべての項目にチェックが入っています。すべて選択、すべての選択をはずす、すべてのレコード および すべてのインデックス のショートカットボタンも使用できます。

テーブルの各行に対して、"アクション" カラムは実行する操作を表示します。テーブルが展開されると、"レコード" および "インデックスフィールド" の行は関連する項目の数を表示します。

"ステータス" カラムは、記号を使用して各項目の検査ステータスを表示します:

	検査の結果、問題はなかった
	検査の結果、問題が見つかった
	検査は部分的に実行された
	検査は実行されなかった

検査を開始するには 検査 ボタンをクリックします。標準ページに戻る時は 標準 ボタンをクリックします。

ログファイルを開く ボタンをクリックすると、マシンのデフォルトブラウザーを使用してログファイルを表示します（上記の [ログファイルを開く](#) 参照）。

標準ページは、詳細ページでおこなわれた変更はまったく考慮しません。標準ページの検査ボタンをクリックすると、すべての項目が検査されます。逆に、詳細ページでおこなわれた設定は、セッションからセッションへと保持されます。

圧縮ページ

このページは、データファイルの圧縮機能にアクセスするときに使用します。

ファイルを圧縮する理由

ファイルの圧縮は以下のニーズに応えるためにおこないます：

- ファイルのサイズの削減と最適化：ファイルには使っていないスペースがあるかもしれません。実際、レコードを削除すると、それらがファイル上で占有していたスペースが空になります。4D はできる限りこういったスペースを再利用しますが、データのサイズは可変なため、連続的に削除や変更をおこなうと、必然的にプログラムにとって使用不可のスペースが作り出されます。大量のデータが削除された直後についても同じことが言えます：空のスペースはそのままファイルに残ります。データファイルのサイズと、実際にデータに使われているスペースの比率をデータの使用率と呼びます。使用率が低すぎると、スペースが無駄なだけではなく、データベースパフォーマンスの低下につながります。圧縮は空きスペースを取り除き、データのストレージを再編成、最適化するためにおこないます。"情報" エリアには、フラグメンテーションに関するデータが要約され、必要な操作が表示されます。MSC の情報ページの [データ](#) タブには、カレントデータファイルのフラグメンテーション情報が表示されます。
- 完全なデータ更新：ストラクチャーファイルの現設定を全データに適用します。同じテーブルのデータが異なる形式で保存されている場合（たとえばデータベースストラクチャーに変更を加えたとき）に便利です。

圧縮はメンテナスマードでのみ可能です。標準モードでこの操作を実行しようとすると、警告ダイアログボックスが表示され、アプリケーションを終了してメンテナスマードで再起動することを知らせます。ただし、アプリケーションが開いていないデータファイルを圧縮することは可能です（[レコードとインデックスを圧縮](#) 参照）。

通常モード

データの圧縮を開始するには、MSC ウィンドウの圧縮ボタンをクリックします。



圧縮はオリジナルファイルのコピーを伴うため、ファイルが格納されているディスクに十分な空きスペースがない場合、ボタンは使用不可になります。

この操作は、メインファイルの他、インデックスファイルもすべて圧縮します。4D はオリジナルファイルをコピーし、オリジナルファイルの隣に作成された Replaced Files (Compacting) フォルダーにそれらを置きます。圧縮操作を複数回実行すると、毎回新しいフォルダーが作成されます。フォルダーネームは、"Replaced Files (Compacting)_1", "Replaced Files (Compacting)_2" のようになります。元のファイルのコピー先は、特殊モードを使って変更できます。

操作が完了すると、圧縮ファイルは自動的にオリジナルファイルと置き換えられます。アプリケーションは即座に操作可能になります。

データベースが暗号化されている場合、復号化と暗号化のステップが圧縮過程に含まれるため、カレントデータの暗号化キーが必要になります。有効なデータキーが未提供の場合には、パスフレーズまたはデータキーを要求するダイアログボックスが表示されます。

警告: 圧縮操作は毎回オリジナルファイルのコピーを伴うため、アプリケーションフォルダーのサイズが大きくなります。アプリケーションのサイズが過剰に増加しないよう、これを考慮することが大切です（とくに、4D アプリケーションがパッケージとして表示される macOS の場合）。パッケージのサイズを小さく保つには、パッケージ内オリジナルファイルのコピーを手動で削除することも役立ちます。

ログファイルを開く

圧縮が完了すると、4D はプロジェクトの Logs フォルダーにログファイルを生成します。このファイルを使用すると実行されたオペレーションをすべて閲覧することができます。このファイルは XML 形式で作成され、*ApplicationName_Compact_Log_yyyy-mm-dd hh-mm-ss.xml* というファイル名がつけられます。

- *ApplicationName* は拡張子を除いたプロジェクトファイルの名前です (例: "Invoices" 等)
- *yyyy-mm-dd hh-mm-ss* はファイルのタイムスタンプです。これはローカルのシステム時間でメンテナンスオペレーションが開始された時刻に基づいています (例: "2019-02-11 15-20-45")。

ログファイルを開くボタンをクリックすると、4Dはマシンのデフォルトブラウザーを使用して直近のログファイルを開きます。

詳細モード

圧縮ページには、データファイル圧縮のオプションページにアクセスするための 特殊 > ボタンがあります。

レコードとインデックスを圧縮

レコードとインデックスを圧縮 には、カレントデータファイルのパス名と、他のデータファイルを指定するのに使用する [...] ボタンが表示されます。このボタンをクリックすると標準のファイルを開くダイアログが表示され、圧縮するデータファイルを選択することができます。開かれているストラクチャーファイルと互換性のあるデータファイルを選択しなければなりません。このダイアログボックスを受け入れると、圧縮するファイルのパス名が更新されます。

2つめの [...] ボタンを使用して、圧縮処理前に元ファイルをコピーする保存先を変更できます。とくに大きなデータファイルを圧縮する際、コピー先を別のディスクに変更するためにこのオプションを使用します。

レコードの強制更新

このオプションが選択されていると、4D は現在のストラクチャー定義に基づき、圧縮処理中に各テーブルのすべてのレコードを再保存します。このオプションが選択されていないと、4D は単にディスク上のデータの並びを再構成するだけです。このオプションは以下のケースで有用です：

- アプリケーションストラクチャーのフィールド型がデータ入力後に変更された場合、たとえば倍長整数型を実数型に変更したようなケースです。4D では (データを失うリスクがあるにしても) まったく異なる型に変更することさえ可能です。たとえば、実数型をテキスト型にすることができます。この場合、4Dは既に入力されたデータを遡及的に変換することはしません。データはレコードがロードされ保存される際に変換されます。このオプションを使用すればデータの変換を強制できます。
- データが入力された後にテキスト、ピクチャー、または BLOB の外部保存オプションが変更された場合。これはとくに v13以前からデータベースを変換した場合に発生します。前述の型変更と同様、4Dはすでに入力されたデータを遡及的に変換しません。入力済みデータに対して新しい保存設定を適用するために、このオプションを選択して圧縮をおこないます。
- テーブルやフィールドが削除された場合。この場合、レコードの再保存をおこないながら圧縮することで、削除された領域を圧縮することができ、ファイルサイズを減らすことができます。

このオプションが選択されていると、すべてのインデックスが更新されます。

アドレステーブル圧縮

(レコードの強制更新を選択した場合にのみ選択可能)

このオプションを使用すると圧縮の際、レコードのアドレステーブルを完全に再構築します。これによりアドレステーブルのサイズが最適化されます。このオプションは主に大量のデータを作成し、そして削除したような場合に使用します。そうでない場合、最適化に明白な意味はありません。

このオプションを使用した場合、圧縮処理に時間がかかるようになり、さらに `SAVE SET` コマンドを使用して保存したセットなど、レコード番号に依存するものが無効になる点に留意してください。そのため、この場合には保存したセットはすべて削除するよう強く推奨します。そうでなければ不正なデータセットを使用することになります。

- 圧縮は、ゴミ箱に入れられたテーブルのレコードも対象とします。ゴミ箱に大量のレコードがある場合、処理が遅くなる原因となります。
- このオプションを使用すると、アドレステーブルは (それに伴ってデータベースそのもの) カレントログファイルとの互換性を失います。ログファイルは自動で保存され、次回アプリケーションを起動した際に新しいログファイルが作成されなければなりません。
- アドレステーブルの圧縮が必要かどうかは、総レコード数と MSC の 情報 ページ内にあるアドレステーブルサイズを比較することで判断することができます。

ロールバックページ

このページは、データファイルに対して実行された操作をロールバックする機能を提供します。この機能は、複数レベルに適用された取り消し機能に似ています。この機能はとくに、間違ってデータベースレコードを削除した場合に便利です。

この機能は、アプリケーションのログファイルが有効なときにのみ使用できます。

Maintenance and Security Center

Rollback

The list below shows all the performed operations recorded in the log file since the last backup.

Operation	Action	Table	Primary...	Process	Size	Date	Hour	System User	4D User	Values	Record
1006	Deletion	Parts	103		63	16/12/2019	18:10	aschmitt	Main user		102
1007	Deletion	Parts	102		63	16/12/2019	18:10	aschmitt	Main user		101
1008	Deletion	Parts	101		63	16/12/2019	18:10	aschmitt	Main user		100
1009	Deletion	Parts	100		63	16/12/2019	18:10	aschmitt	Main user		99
1010	Deletion	Parts	99		63	16/12/2019	18:10	aschmitt	Main user		98
1011	Deletion	Parts	98		63	16/12/2019	18:10	aschmitt	Main user		97
1012	Deletion	Parts	97		63	16/12/2019	18:10	aschmitt	Main user		96
1013	Deletion	Parts	146		63	16/12/2019	18:10	aschmitt	Main user		145
1014	Deletion	Parts	145		63	16/12/2019	18:10	aschmitt	Main user		144
1015	Deletion	Parts	144		63	16/12/2019	18:10	aschmitt	Main user		143
1016	Deletion	Parts	143		63	16/12/2019	18:10	aschmitt	Main user		142
1017	Deletion	Parts	142		63	16/12/2019	18:10	aschmitt	Main user		141
1018	Deletion	Parts	141		63	16/12/2019	18:10	aschmitt	Main user		140
1019	Deletion	Parts	8		63	16/12/2019	18:10	aschmitt	Main user		7
1020	Deletion	Parts	7		63	16/12/2019	18:10	aschmitt	Main user		6
1021	Deletion	Parts	6		63	16/12/2019	18:10	aschmitt	Main user		5

By selecting a specific log action in the list above and clicking the Rollback button, 4D Application will close the current data file, restore and open the selected backup of the database and perform all the above logged actions including the selected one.

The rollback operation cannot be undone, but the current data file will be renamed and stored on the disk.

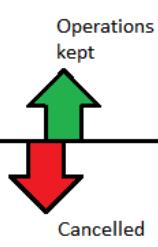
Current log file

Rollback

データベースが暗号化されており、開かれたログファイルに対応する有効なデータキーが提供されていない場合、暗号化された値は 値 カラムには表示されません。そのような状況で ロールバック ボタンをクリックすると、パスフレーズまたはデータキーを要求するダイアログボックスが表示されます。

操作リストの内容と動作は [ログ解析](#) ページのものと同じです。

操作のロールバックをおこなうには、それ以降の全操作をキャンセルする行を選択します。選択された行が保持される最後の操作になります。たとえば削除をキャンセルしたい場合、その削除操作のひとつ前の行を選択します。すると、削除操作とそれ以降の処理がすべてキャンセルされます。



The diagram illustrates a table of database operations. A green arrow points upwards from the table, labeled "Operations kept", indicating that these operations are retained. A red arrow points downwards, labeled "Cancelled operations", indicating that these operations are discarded. A specific row in the table is highlighted in blue, and a cell within that row is highlighted in yellow, likely representing the selected operation for rollback.

Operation	Action	Table	Primary...	Process	Size	Date	Hour	System User	4D User	Values	Record
1006	Deletion	Parts	103		63	16/12/2019	18:10	aschmitt	Main user		102
1007	Deletion	Parts	102		63	16/12/2019	18:10	aschmitt	Main user		101
1008	Deletion	Parts	101		63	16/12/2019	18:10	aschmitt	Main user		100
1009	Deletion	Parts	100		63	16/12/2019	18:10	aschmitt	Main user		99
1010	Deletion	Parts	99		63	16/12/2019	18:10	aschmitt	Main user		98
1011	Deletion	Parts	98		63	16/12/2019	18:10	aschmitt	Main user		97
1012	Deletion	Parts	97		63	16/12/2019	18:10	aschmitt	Main user		96
1013	Deletion	Parts	146		63	16/12/2019	18:10	aschmitt	Main user		145
Selected operation	1014	Deletion	Parts	145	63	16/12/2019	18:10	aschmitt	Main user	144	
	1015	Deletion	Parts	144	63	16/12/2019	18:10	aschmitt	Main user	143	
	1016	Deletion	Parts	143	63	16/12/2019	18:10	aschmitt	Main user	142	
	1017	Deletion	Parts	142	63	16/12/2019	18:10	aschmitt	Main user	141	
	1018	Deletion	Parts	141	63	16/12/2019	18:10	aschmitt	Main user	140	
	1019	Deletion	Parts	8	63	16/12/2019	18:10	aschmitt	Main user	7	
	1020	Deletion	Parts	7	63	16/12/2019	18:10	aschmitt	Main user	6	
	1021	Deletion	Parts	6	63	16/12/2019	18:10	aschmitt	Main user	5	

次に、ロールバック ボタンをクリックします。4Dは処理を続行してもよいか、確認してきます。OK をクリックすると、データは選択された行の状態に戻ります。

アーカイブされたファイルから復旧したデータベースに対してロールバックをおこなう場合には、ウィンドウの下部にあるメニューを使用して適用するログファイルを選択できます。この場合、アーカイブに対応するログファイルを指定しなければなりません。

ロールバックは次のように動作します：ユーザーが ロールバック ボタンをクリックすると、4Dはカレントデータベースを閉じ、最新のバックアップからデータベースの復元をおこないます。復元されたデータベースが開かれ、4Dはログファイル中で選択された操作までを統合します。データベースがまだ保存されていない場合、4Dは空のデータファイルを使います。

修復ページ

このページは、データファイルが損傷を受けたとき、それを修復するために使用します。一般的にこれらの機能は、4Dの起動時や MSC による 検査 の結果アプリケーションに損傷を見つけたときに、4D技術者の監督下で実行します。

警告: 修復操作は毎回オリジナルファイルのコピーを伴うため、アプリケーションフォルダーのサイズが大きくなります。アプリケーションのサイズが過剰に増加しないよう、これを考慮することが大切です（とくに、4Dアプリケーションがパッケージとして表示される macOS の場合）。パッケージのサイズを小さく保つには、パッケージ内オリジナルファイルのコピーを手動で削除することも役立ちます。

修復はメンテナンスマードでのみ可能です。標準モードでこの操作を実行しようとすると、警告ダイアログが表示され、アプリケーションを終了してメンテナンスマードで再起動することを知らせます。データベースが暗号化されている場合、復号化と暗号化的ステップが修復過程に含まれるため、カレントデータの暗号化キーが必要になります。有効な暗号化キーがまだ提供されていない場合、パスフレーズ、あるいは暗号化キーを要求するダイアログが表示されます（暗号化ページ参照）。

データファイル修復

修復するデータ

カレントデータファイルのパス名。 [...] ボタンを使って、他のデータファイルを指定することができます。このボタンをクリックすると標準のファイルを開くダイアログが表示され、修復するデータファイルを選択することができます。標準の修復 を実行する場合、開かれたストラクチャーに対応するデータファイルを選択しなければなりません。レコードヘッダーによる再生 を実行する場合、どのデータファイルでも選択できます。このダイアログを受け入れると、ウィンドウには修復対象のファイルのパス名が表示されます。

オリジナルをここに移動

デフォルトで修復処理の前に元のデータファイルが複製されます。このファイルは、アプリケーションフォルダーの "Replaced files (repairing)" サブフォルダーに配置されます。二つ目の [...] ボタンを使用して、複製ファイルの保存先を変更できます。とくに大きなデータファイルを修復する際、コピー先を別のディスクに変更するためにこのオプションを使用します。

修復済ファイル

4Dは元のファイルの場所に空のデータファイルを新規作成します。元のファイルは、"¥Replaced Files (Repairing) {日付} {時刻}"という名前のフォルダーがオリジナルの移動先に指定したフォルダー内に作成され（デフォルトはアプリケーションフォルダー）、そこに移動されます。修復されたデータは、新規作成された空のファイルに格納されます。

標準の修復

少数のレコードやインデックスが損傷を受けているケース（アドレステーブルは損傷を受けていない）では、標準の修復を選択します。データは圧縮および修復されます。このタイプの修復をおこなうには、データファイルとストラクチャーファイルが対応していなければなりません。

修復操作が完了すると、MSCの "修復" ページが表示され、修復の結果が表示されます。修復に成功していれば、その旨のメッセージが表示されます。その場合、アプリケーションをそのまま使い始めることができます。



レコードヘッダーによる再生

このローレベルな修復オプションは、データファイルが大きく損傷していて、バックアップの復元や標準の修復など、他の手段では回復できなかった場合にのみ使用します。

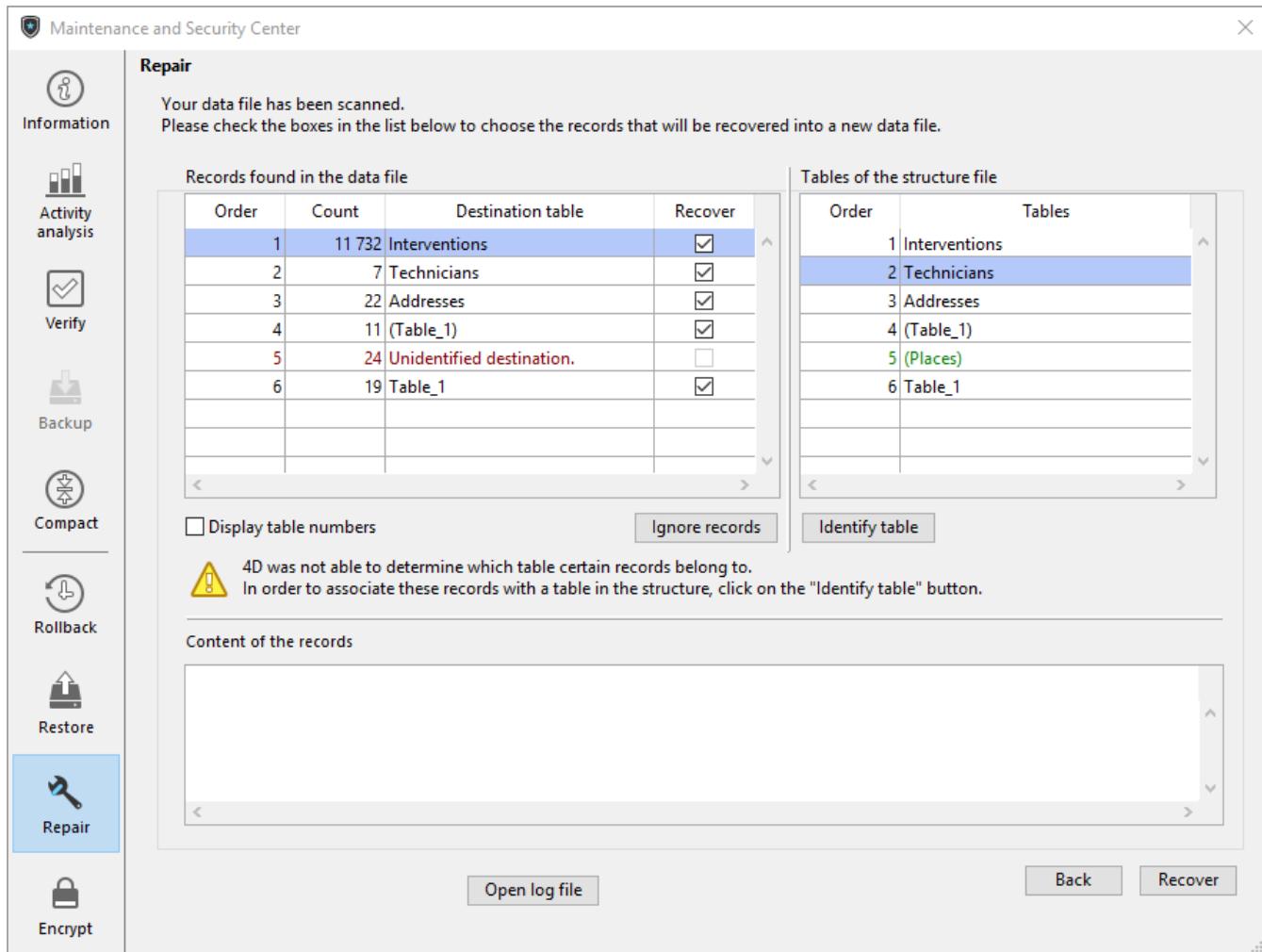
4Dのレコードはサイズが可変です。故に、そのレコードをロードするには、ディスク上のどこに格納されているか、その場所を "アドレステーブル" という専用テーブルに記録しておく必要があります。プログラムは、インデックスやアドレステーブルを経由して、レコードのアドレスにアクセスします。レコードやインデックスのみが損傷を受けている場合、標準の修復を使用すれば通常は問題が解決されます。しかし、アドレステーブル自身が損傷を受けている場合には、これを再構築しなくてはならないため、より高度な修復作業が必要となります。これをおこなうために、MSC は各レコードのヘッダーに位置するマーカーを使

用します。マーカーがレコード情報のサマリーと比較されることにより、アドレステーブルが再構築可能となります。

ストラクチャーのテーブルプロパティで レコードを完全に削除 オプションを解除していると、ヘッダーマーカーを使用した復旧によって削除したはずのレコードが復活する原因となります。

ヘッダーによる再生において、整合性の制約は考慮されません。この処理をおこなった後、重複不可フィールドに重複する値が現れたり、NULL 値を許可しない に定義したフィールドに NULL 値が現れたりするかもしれません。

スキャンおよび修復... ボタンをクリックすると、4Dはデータファイルを完全にスキャンします。スキャンを完了すると、結果が以下のウィンドウに表示されます:



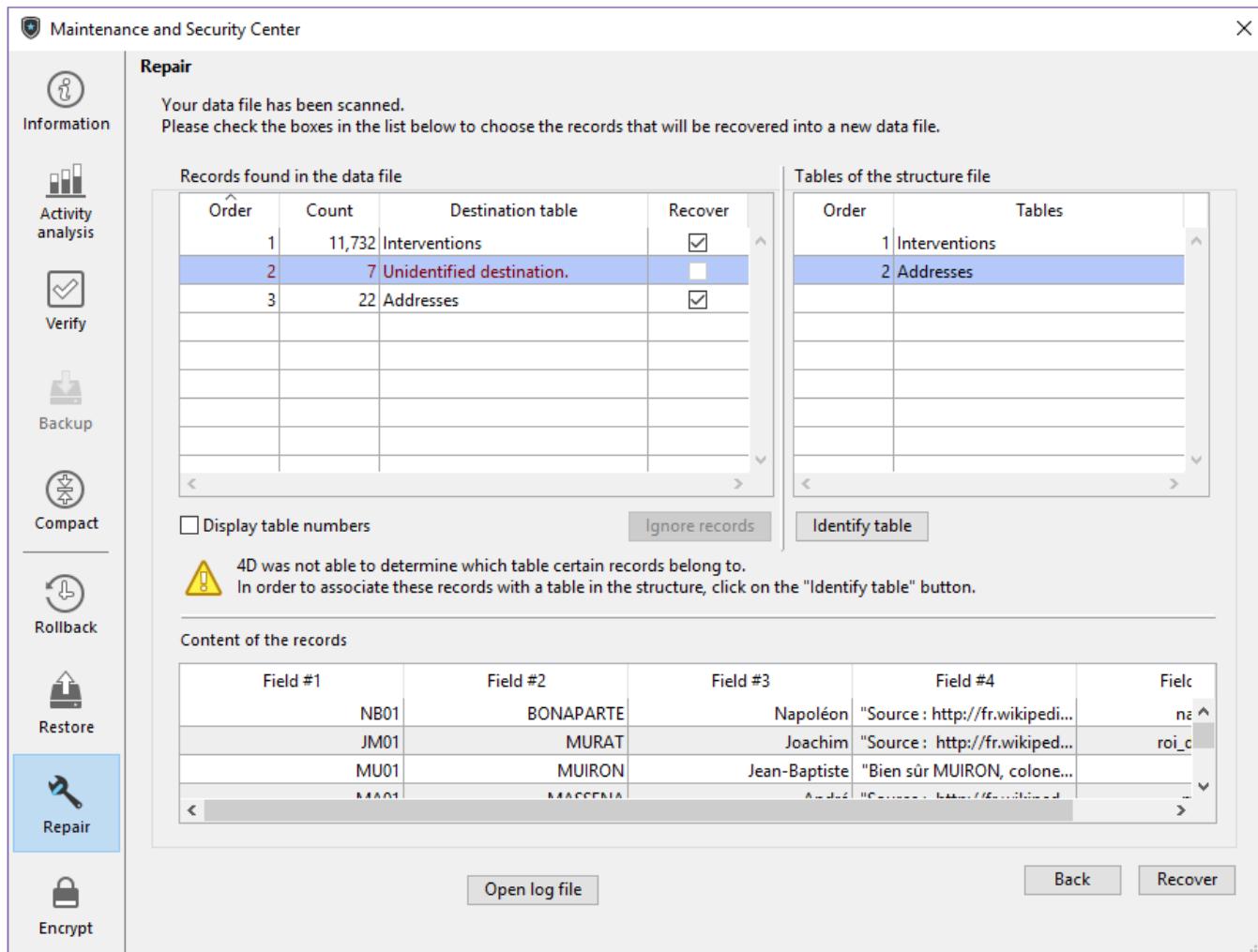
すべてのレコードおよびすべてのテーブルに割当先が見つかった場合は、メインエリアのみが表示されます。

- "データファイル中で見つかったレコード" エリアには 2つのリストがあり、データスキヤン結果の概要が表示されます。
- 左のリストには、データファイルスキヤンの情報が表示されます。各行には、データファイル中の再生可能なレコードのグループが表示されます:
 - 順番 の列には、レコードグループの再生順が表示されます。
 - カウント 列には、グループに含まれるレコード数が表示されます。
 - 割当先テーブル 列には、識別されたレコードのグループに割り当てられたテーブルの名前が表示されます。割り当てられたテーブルの名前は自動で緑色で表示されます。割り当てられなかったグループ、つまりどのレコードにも関連づけることができなかったテーブルは赤色で表示されます。
 - 再生 列では、レコードを再生するかどうかを各グループごとに指定できます。デフォルトで、テーブルに割り当てられるすべてのグループが選択されています。
 - 右側のリストには、プロジェクトファイルのテーブルが表示されます。

手動による割り当て

アドレステーブルが損傷を受けているため、テーブルに割り当てるうことのできないレコードグループがある場合、それらを手動で割り当てることができます。これにはまず、左側のリストの中で割り当てられていないレコードグループを選択します。グループ先頭の複数レコードの内容が "レコードの内容" エリアにプレ

ビューされるため、それがどのテーブルのレコードか判断しやすくなります:



次に "割り当てられていないテーブル" リストから、グループを割り当てるテーブルを選択し、テーブルを識別 ボタンをクリックします。割り当てのためにドラッグ & ドロップを使用することもできます。結果そのレコードグループは選択したテーブルに割り当てられ、そのテーブルのレコードとして再生されます。手動で割り当てられたテーブルはリスト中黒色で表示されます。レコードを無視する ボタンをクリックすると、レコードグループに対するテーブルの割り当てを手動で解除できます。

ログファイルを開く

修復が完了すると、4D はプロジェクトの Logs フォルダーにログファイルを生成します。このファイルを使用すると実行されたオペレーションをすべて閲覧することができます。このファイルは XML 形式で作成され、*ApplicationName_Repair_Log_yyyy-mm-dd hh-mm-ss.xml* というファイル名がつけられます。

- *ApplicationName* は拡張子を除いたプロジェクトファイルの名前です (例: "Invoices" 等)
- *yyyy-mm-dd hh-mm-ss* はファイルのタイムスタンプです。これはローカルのシステム時間でメンテナンスオペレーションが開始された時刻に基づいています (例: "2019-02-11 15-20-45")。

ログファイルを開く ボタンをクリックすると、4D はマシンのデフォルトブラウザーを使用して直近のログファイルを開きます。

暗号化ページ

このページを使用して、データベースの各テーブルに対して定義された 暗号化可能 属性に基づいて、データファイルを暗号化または 復号化（つまりデータから暗号化を解除）することができます。

4D のデータ暗号化についての詳細な情報に関しては、デザインリファレンス マニュアルの [データの暗号化](#) の章を参照してください。また、[A deeper look into 4D data encryption](#) のブログ記事（英文）も参照ください。

暗号化/復号化操作をおこなうたびに、新しいフォルダーが作成されます。そのフォルダーは "Replaced Files (Encrypting) yyyy-mm-dd hh-mm-ss" あるいは "Replaced Files (Decrypting) yyyy-mm-dd hh-mm-ss" と名前が付けられます。

暗号化は [メンテナスマード](#) でのみ利用可能です。標準モードでこの操作を実行しようとすると、警告ダイアログが表示され、アプリケーションを終了してメンテナスマードで再起動することを知らせます。

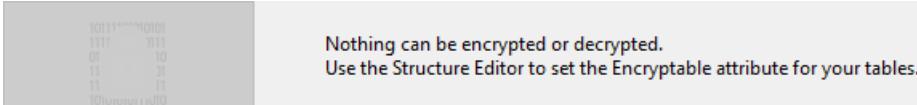
警告:

- データファイルの暗号化は時間がかかる操作です。実行中は（ユーザーによって割り込み可能な）進捗インジケーターが表示されます。また、アプリケーションの暗号化操作には必ず圧縮のステップが含まれるという点に注意してください。
- 暗号化操作をおこなうたびに、その操作はデータファイルのコピーを作成し、その結果アプリケーションファイルのサイズは増大します。アプリケーションのサイズが過剰に増加しないよう、これを考慮することが大切です（とくに、4D アプリケーションがパッケージとして表示される macOS の場合）。パッケージのサイズを小さく保つには、パッケージ内オリジナルファイルのコピーを手動で削除/移動することも役立ちます。

データを初めて暗号化する場合

MSC でデータファイルを初めて暗号化する場合、以下のような手順を踏む必要があります：

- ストラクチャーエディターにおいて、データを暗号化したいテーブルに対して 暗号化可能 属性にチェックを入れます。詳細は "テーブルプロパティ" の章を参照してください。
- MSC の暗号化ページを開きます。どのテーブルにも 暗号化可能 属性を付けないでページを開こうとした場合、ページに以下のメッセージが表示されます：



そうでない場合には、以下のメッセージが表示されます：

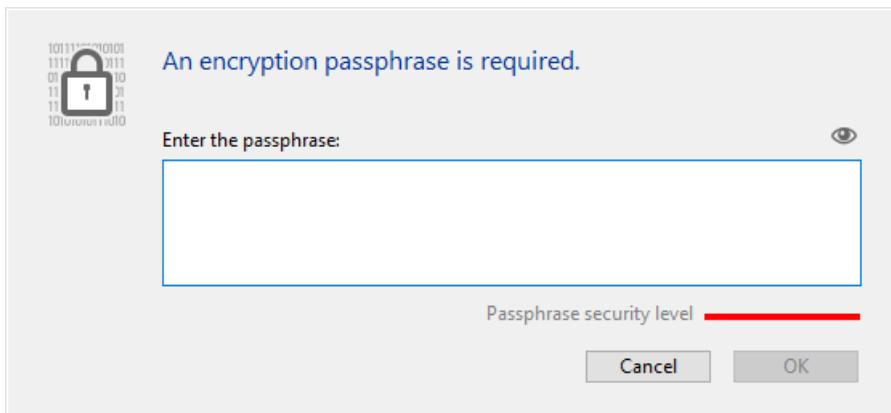


このメッセージは、少なくとも 1 つのテーブルに対して 暗号化可能 属性のステータスが変更されていて、データファイルがまだ暗号化されていないことを意味します。注：すでに暗号化されているデータファイル、または復号化されたデータファイルに対して、暗号化可能 属性のステータスが変更された場合にも同じメッセージが表示されます（以下参照）。

- 暗号化ピクチャーボタンをクリックします。



データファイル用のパスフレーズを入力するように聞かれます：



パスフレーズはデータ暗号化キーを生成するのに使用されます。パスフレーズはパスワードの強化版のようなもので、大量の文字を含めることができます。たとえば、"We all came out to Montreux" あるいは "My 1st Great Passphrase!!" のようなパスフレーズを入力することができます。

パスフレーズの安全性は、セキュリティレベルインジケーターによって確認できます: Passphrase security level: (濃い緑色がもっとも安全なレベルであることを示します)。

4. Enter を押して安全なパスフレーズの入力を確定します。

暗号化プロセスがスタートします。MSC が標準モードで開かれていた場合、アプリケーションはメンテナンスマードで再起動されます。

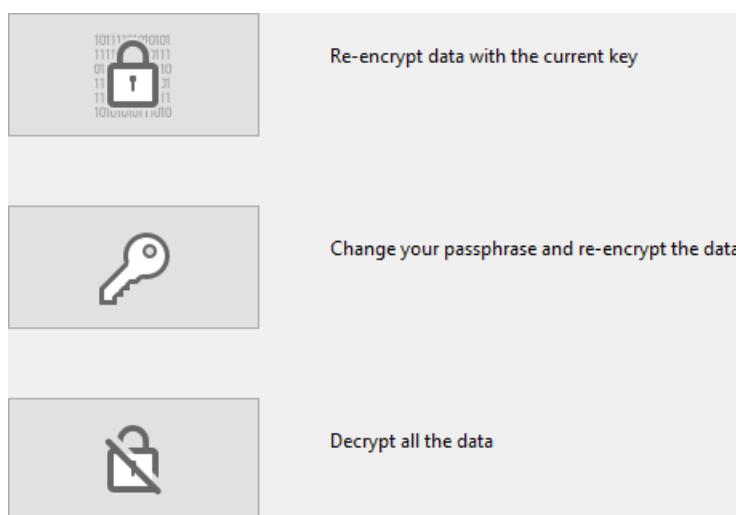
4D では暗号化キーを保存することができます (以下の [暗号化キーを保存する](#) の段落を参照してください)。暗号化キーの保存は、このタイミングか、あるいは後でおこなうこともできます。また暗号化ログファイルを開くこともできます。

暗号化プロセスが正常に完了した場合、暗号化ページは [暗号化メンテナンスオペレーション](#) ボタンを表示します。

警告: 暗号化操作の最中、4D は新しい、空のデータファイルを作成したうえで、元のデータファイルからデータを注入します。"暗号化可能" テーブルに属しているレコードは暗号化後にコピーされ、他のレコードは単にコピーされるだけです (圧縮オペレーションも実行されます)。操作が正常に完了した場合、もとのデータファイルは "Replaced Files (Encrypting)" フォルダーへ移動されます。暗号化されたデータファイルを配布する場合、暗号化されていないデータファイルをアプリケーションフォルダーからすべて移動/削除しておくようにしてください。

暗号化メンテナンスオペレーション

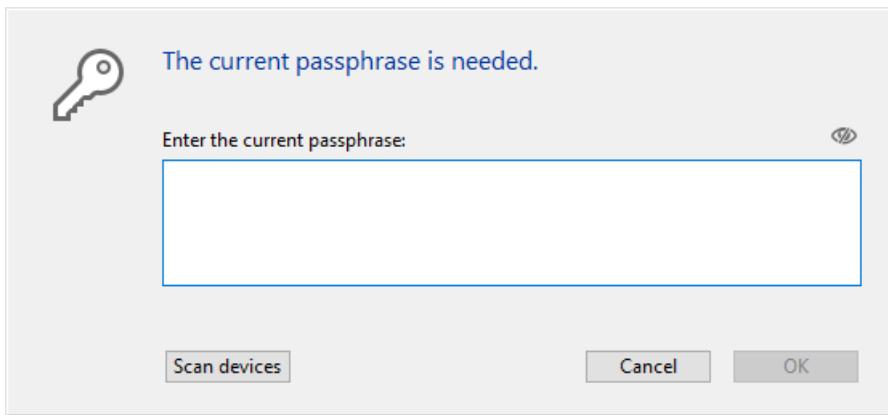
アプリケーションが暗号化されているとき (上記参照)、暗号化ページでは、標準のシナリオに対応した様々な暗号化メンテナンスオペレーションを提供します。



カレントの暗号化キーを入力する

セキュリティ上の理由から、すべての暗号化メンテナンスオペレーションはカレントのデータ暗号化キーの入力を要求します。

- データ暗号化キーが既に 4Dキーチェーン (1) に読み込まれている場合、そのキーは 4D によって自動的に再利用されます。
- データ暗号化キーが見つからない場合、それを入力する必要があります。以下のようなダイアログが表示されます:



この段階では 2つの選択肢があります:

- カレントのパスフレーズ (2) を入力し、OK をクリックする。OR
- USBキーなどのデバイスを接続して、デバイスをスキャンボタンをクリックする。

(1) 4Dキーチェーンは、アプリケーションのセッション中に入力されたすべての有効なデータ暗号化キーを保管します。
(2) カレントのパスフレーズとは、カレントのデータ暗号化キーを生成するのに使用されたパスフレーズです。

いずれの場合においても、有効なパスフレーズ/暗号化キーが提供されると、4D は（まだメンテナスマードではなかった場合は）メンテナスマードで再起動し、選択されたオペレーションを実行します。

カレントの暗号化キーでデータを再暗号化する

この操作は、データを格納している 1つ以上のテーブルにおいて 暗号化可能 属性が変更された場合に有用です。この場合、データの整合性を保つために、4D はアプリケーション内のそのテーブルのレコードへの書き込みアクセスを禁止します。有効な暗号化ステータスを得るために、データの再暗号化が必要になります。

1. カレントの暗号化キーでデータを再暗号化 をクリックします。
2. カレントのデータ暗号化キーを入力します。

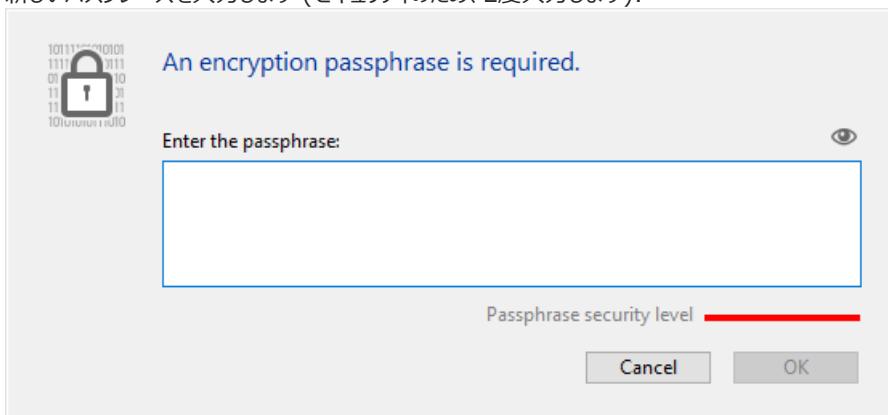
データファイルはカレントのデータ暗号化キーで正常に再暗号化され、確認メッセージが表示されます:



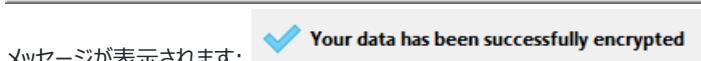
パスフレーズを変更してデータを再暗号化する

この操作は、カレントの暗号化データキーを変更したい場合に有用です。たとえば、セキュリティ上のルール（3ヶ月ごとにパスフレーズを変更する必要があるなど）を遵守するために変更をおこないたいケースが考えられます。

1. パスフレーズを変更してデータを再暗号化する をクリックします。
2. カレントのデータ暗号化キーを入力します。
3. 新しいパスフレーズを入力します（セキュリティのため、2度入力します）:



データファイルは新しいキーで暗号化され、確認



全データを復号化

この操作は、データファイルからすべての暗号化を取り除きます。データを暗号化しておきたくない場合、以下の手順に従ってください：

1. 全データを復号化 をクリックします。
2. カレントのデータ暗号化キーを入力します ([カレントの暗号化キーを入力する 参照](#))。

データは完全に復号化され、確認メッセージが表示されます：



データファイルが復号化されると、テーブルの暗号化ステータスは暗号化可能属性と合致しなくなります。ステータスを合致させるためには、データベースのストラクチャレベルにおいてすべての 暗号化可能 属性を選択解除しなければなりません。

暗号化キーを保存する

4D ではデータ暗号化キーを専用ファイルに保存しておくことができます。このファイルを USBキーなどの外部デバイスに保存しておくと、暗号化されたアプリケーションを使うのが簡単になります。なぜならユーザーは暗号化されたデータにアクセスするには、アプリケーションを開く前にデバイスを接続してキーを提供すればよいからです。

新しいパスフレーズが提供されるたびに暗号化キーを保存することができます：

- アプリケーションが最初に暗号化されたとき
- アプリケーションが新しいパスフレーズで再暗号化されたとき

連続した暗号化キーを同じデバイスに保存することが可能です。

ログファイル

暗号化オペレーションが完了すると、4D はアプリケーションの Logs フォルダー内にファイルを生成します。このファイルは XML 形式で作成され、"ApplicationName_Encrypt_Log_yyyy-mm-dd hh-mm-ss.xml" または "ApplicationName_Decrypt_Log_yyyy-mm-dd hh-mm-ss.xml" という名前がつけられます。

新しくログファイルが生成されるたび、MSC ページに ログファイルを開く ボタンが表示されます。

このログファイルには、暗号化/復号化プロセスの間に実行された内部オペレーションがすべて記録されているほか、エラー（あれば）が記録されています。

バックアップ

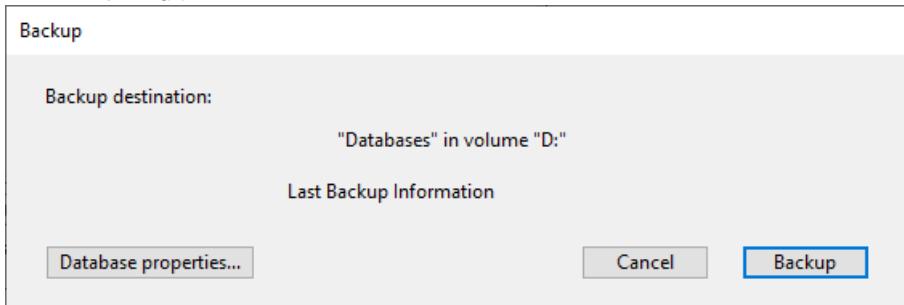
4D では、次の 3つの方法でバックアップを開始することができます:

- 手動による方法: 4D の ファイル メニューから **バックアップ...** コマンドを使用します。または、[Maintenance & Security Center \(MSC\)](#) の バックアップ ボタンをクリックします。
- 自動的に行う方法: ストラクチャー設定からスケジューラーを使用します。
- プログラムによる方法: **BACKUP** コマンドを使用します。

4D Server: リモートマシンから **BACKUP** を呼び出すメソッドを使用して、手動でバックアップを開始することができます。いかなる場合でも、このコマンドはサーバー上で実行されます。

手動バックアップ

- 4D の ファイル メニューから **バックアップ...** コマンドを選択します。



バックアップウインドウが表示されます: "バックアップファイルの保存先" エリアの隣のポップアップメニューを使用して、バックアップファイルの保存場所を確認することができます。この場所はデータベース設定の バックアップ/設定 ページにて、設定されています。

- 4D の [Maintenance & Security Center \(MSC\)](#) を開いて、[バックアップページ](#) から操作することもできます。

データベースプロパティ... ボタンをクリックすると、ストラクチャー設定のバックアップ/設定ページが表示されます。

- バックアップ をクリックし、現在のパラメーターを用いてバックアップを開始します。

定期的な自動バックアップ

自動バックアップは指定されたスケジュールに基づいて自動的に実行されます。バックアップの周期は、ストラクチャー設定 の バックアップ/スケジューラー ページにて設定します。

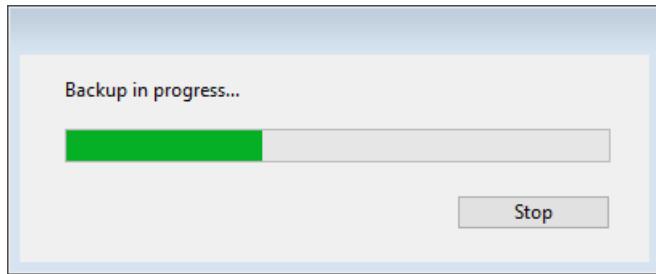
ユーザーが一切操作しなくても、このページで指定された時間にバックアップが自動実行されます。このダイアログボックスに関する詳細は、[バックアップ設定のスケジューラー](#) の項目を参照してください。

BACKUP コマンド

任意のメソッドにて **BACKUP** 4Dランゲージコマンドを実行すると、ストラクチャー設定に定義されている現在のパラメーターを用いてバックアップを開始します。バックアッププロセスを処理するため、**On Backup Startup** および **On Backup Shutdown** データベースメソッド使用することができます (詳細は、[4Dランゲージリファレンス マニュアル](#)を参照ください)。

バックアップ処理の管理

バックアップが開始すると 4Dは、バックアップの進捗状況を知らせるサーモメーターのあるダイアログボックスを表示します:



MSC を使用している場合、この進捗インジケーターは [MSC のバックアップページ](#) に表示されます。

中止 ボタンをクリックすると、いつでもバックアップを中断することができます ([バックアップ中に問題が発生した場合 参照](#))。

前回のバックアップの結果 (成功または不成功) は、[MSC のバックアップページ](#) の "前回のバックアップの情報" エリア、または 4D Server の メンテナンスページ で確認できます。また、データベースの バックアップジャーナル (Backup Journal.txt) にも記録されます。

バックアップ中のアプリケーションへのアクセス

バックアップ実行中のアプリケーションへのアクセスは、4D によって制限されます。4D は、バックアップに含まれたファイルタイプに関連するプロセルはすべてロックします：プロジェクトファイルだけがバックアップされている場合、ストラクチャーにはアクセスできませんが、データにはアクセス可能です。

反対に、データだけがバックアップされているのであれば、ストラクチャーへのアクセスは許可されます。この場合に、アプリケーションへのアクセスが可能かどうかを次に示します：

- シングルユーザー版の 4D の場合、アプリケーションは読み込み、書き込みともにロックされ、すべてのプロセスが停止します。実行できるアクションはありません。
- 4D Server の場合、アプリケーションへの書き込みだけがロックされ、クライアントマシンはデータを照会することができます。クライアントマシンからサーバーへ追加・削除・変更のリクエストが送信されると、ウインドウが表示され、バックアップの終了まで待機するよう要求されます。アプリケーションが保存されるとウインドウが閉じられ、要求したアクションが実行されます。バックアップの終了まで待機せずに、処理中のリクエストをキャンセルするには、処理をキャンセル ボタンをクリックします。ただし、バックアップ前に開始したメソッドから要求されたアクションが実行待機中である場合、このアクションをキャンセルすべきではありません。この場合、実行すべき残りの処理だけがキャンセルされてしまうためです。しかも、メソッドの一部は実行済みなので、データにおいて論理上の不整合が生じる可能性があります。> 実行待機中のアクションが、メソッドから要求されたものである場合に、ユーザーが処理をキャンセル ボタンをクリックすると、4D Server はエラー -9976 (データベースのバックアップが進行中なので、このコマンドは実行されません) を返します。

バックアップ中に問題が発生した場合

バックアップが正常に実行されない場合もあります。バックアップが不成功に終わる原因としては、ユーザーによる中断、添付ファイルが見つからない場合、保存先ディスクのトラブル、不完全なトランザクションなど、いくつか考えられます。4D は原因に応じて問題に対処します。

すべての場合において、前回のバックアップのステータス (成功または不成功) は、[MSC のバックアップページ](#) の "前回のバックアップの情報" エリア、4D Server の メンテナンスページ、および バックアップジャーナル (Backup Journal.txt) に表示されます。

- ユーザーによる中断：進捗ダイアログボックスの 中止 ボタンをクリックすると、いつでもバックアップを中断することができます。この場合、各項目のコピーが中止されてエラー 1406 が生成されます。このエラーは On Backup Shutdown データベースメソッドで遮ることができます。
- 添付ファイルが見つからない：添付ファイルが見つからない場合、4D はバックアップを部分的に実行し (アプリケーションファイルおよびアクセス可能な添付ファイルのバックアップ)、エラーを返します。
- バックアップ不可能 (ディスクフル、ディスクの書き込み保護、ディスクが見つからない、ディスク障害、不完全なトランザクション、定期的な自動バックアップ時にアプリケーションが起動されていないなど)：初回のエラーの場合、4D はもう一度バックアップの実行を試みます。この 2 回のバックアップ間の待機時間は、ストラクチャー設定の バックアップ/バックアップ & 復旧 ページで指定します。再試行にも失敗した場合、システムの警告ダイアログボックスが表示されてエラーが生成されます。このエラーは On Backup Shutdown データベースメソッドで遮ることができます。

バックアップジャーナル

バックアップの追跡や検証を容易にするため、バックアップモジュールは実行された各処理の概要を特別なファイルに書き込みます。このファイルは、いわゆる活動記録のようなものです。処理が定期的または手動のいずれでおこなわれていても、すべてのデータベース操作 (バックアップ、復元、ログファイルの統合) がこのファイルに、日誌のごとく記録されます。これらの処理が実行された日付と時刻もこのジャーナルに記述されます。

バックアップジャーナルには "Backup Journal[001].txt" という名前が付けられ、プロジェクトの "Logs" フォルダーに配置されます。バックアップジャーナルは、任意のテキストエディターで開くことができます。

バックアップジャーナルのサイズ管理

バックアップ方法によっては、バックアップジャーナルのサイズがすぐに大きくなってしまうことがあります（たとえば、添付ファイルと一緒にバックアップされる場合）。このサイズを管理するには、2つの方法があります：

- **自動バックアップ**: 4D はバックアップを実行する前にカレントバックアップジャーナルファイルのサイズを確認します。10MB よりも大きい場合、カレントファイルはアーカイブされ、[xxx] の番号がインクリメントされた新しいファイルを作成します（例: "Backup Journal[002].txt"）。ファイル番号が 999 を超えると、ナンバリングは 1 に戻り、既存ファイルが置換されます。
- **記録する情報量を削減する**: このためには、プロジェクトの *backup.4DSettings* ファイルの *VerboseMode* キーの値を変更します。デフォルトでは、*true* の値が設定されています。この値を *false* に変更すると、バックアップジャーナルには主要な情報のみが記録されます（スタート時の日付と時刻、そしてエラーの有無）。バックアップ設定に使われる XMLキーについての説明は [バックアップ設定ファイル](#) マニュアルを参照ください。

backupHistory.json

最新のバックアップと復元処理についての情報はすべて、アプリケーションの *backupHistory.json* ファイルに記録されます。記録されるのは、保存されたファイル（添付含む）のパスのほか、回数、日付、時刻、所要時間、各処理のステータスです。ファイルサイズを制限するため、バックアップ＆復旧ページの一般設定にある "最新のバックアップのみ保存 X バックアップファイル" に指定した数と同じ分だけ、処理のログを保持します。

backupHistory.json ファイルはカレントのバックアップ保存先フォルダーに作成されます。以下のコードを実行することで、このファイルの実際のパスを取得することができます：

```
$backupHistory:=Get 4D file(Backup history file)
```

警告

backupHistory.json ファイルを削除または移動した場合、次のバックアップ番号はリセットされるという点に注意してください。

backupHistory.json ファイルは 4D 用にフォーマットされています。バックアップ処理のレポートを直接読んで確認するには、バックアップジャーナルの方が適切です。

バックアップ設定

バックアップ設定の定義は、[ストラクチャー設定ダイアログボックス](#) 内で 3ページにわたっています。次の設定がおこなえます:

- 自動バックアップ用のスケジューラー設定
- 各バックアップに含めるファイル
- 自動タスクの実行を可能にする高度な設定

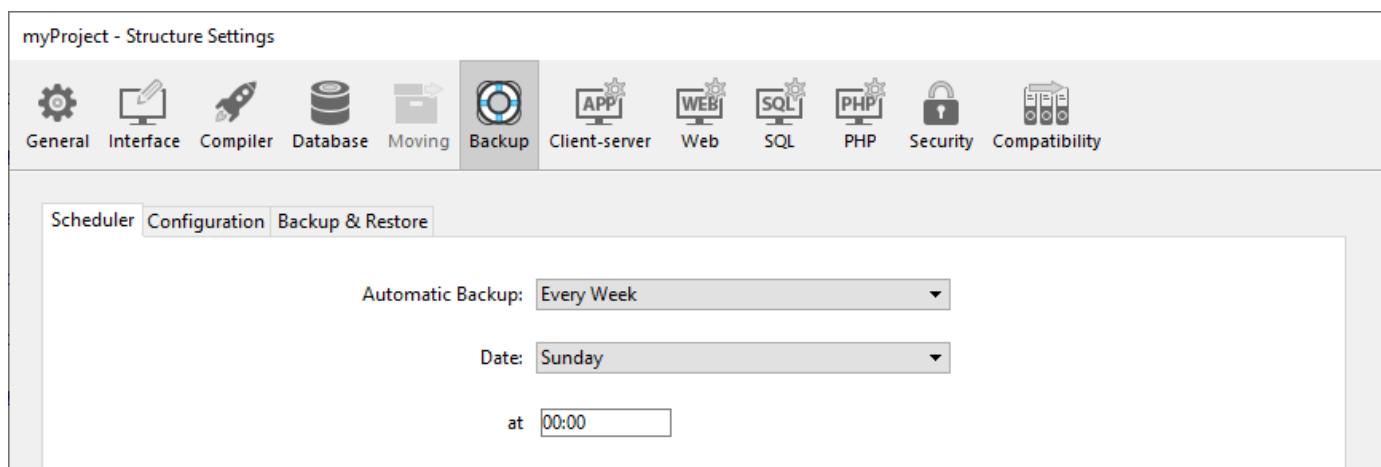
このダイアログボックスで定義された設定は *Backup.4DSettings* ファイルに書き込まれ、[Settings フォルダー](#) に保存されます。

スケジューラー

4D や 4D Server で開かれているアプリケーションのバックアップを自動化することができます（クライアントマシンが接続されている必要はありません）。これはバックアップ周期（時間、日、週、月単位等）を設定することによりおこないます。現在のバックアップ設定に基づき、4D は自動でバックアップを実行します。

バックアップが実行されるべきときにアプリケーションが起動されていなかった場合には、次に起動されたとき 4D はバックアップが失敗したものと認識し、ストラクチャー設定の再試行設定を適用します（[バックアップ中に問題が発生した場合](#) 参照）。

バックアップのスケジュール設定は、ストラクチャー設定の [バックアップ/スケジューラー](#) ページでおこないます：



The screenshot shows the 'myProject - Structure Settings' window. The 'Backup' tab is selected. Under the 'Scheduler' tab, the 'Automatic Backup' dropdown is set to 'Every Week'. The 'Date:' dropdown is set to 'Sunday' and the 'at' dropdown is set to '00:00'.

このページにあるオプションを使用して、アプリケーションの自動バックアップのスケジュールを設定できます。標準のクリック設定、または完全なカスタマイズを選択できます。自動バックアップ メニューでの選択に基づき、さまざまなオプションが表示されます：

- しない: スケジュールに基づくバックアップは無効となります。
- 毎時: 次の時間以降、毎時間ごとに自動バックアップをおこないます。
- 毎日: 日に一回自動バックアップをおこないます。バックアップを何時に開始するかを設定します。
- 毎週: 週に一回自動バックアップをおこないます。バックアップを開始する曜日と時刻を入力するエリアが表示されます。
- 毎月: 月に一回自動バックアップをおこないます。バックアップを開始する日付と時刻を入力するエリアが表示されます。
- カスタマイズ: 自動バックアップを詳細にスケジュールする場合に使用します。このオプションを選択すると、複数の入力エリアが表示されます：
 - X 時間ごと: 時間単位でバックアップの間隔をスケジュールできます。1から24までの値を設定できます。
 - X 日ごと: 日単位でバックアップの間隔をスケジュールできます。たとえば、毎日バックアップをおこなうには 1 と設定します。このオプションを選択した場合、バックアップが開始される時刻を設定しなければなりません。
 - X 週ごと: 週単位でバックアップの間隔をスケジュールできます。たとえば、毎週バックアップをおこなうには 1 と設定します。このオプションを選択した場合、バックアップを開始する曜日と時刻を設定しなければなりません。複数の曜日を選択することもできます。たとえば、毎週水曜日と金曜日にバックアップをするようプログラムできます。
 - X 月ごと: 月単位でバックアップの間隔をスケジュールできます。たとえば、毎月バックアップをおこなうには 1 と設定します。このオプションを選択した場合、バックアップを開始する日付と時刻を設定しなければなりません。

夏時間と標準時の切り替えがある場合にはスケジューラーが一時的に影響され、次のバックアップ実行が 1時間ずれる場合があります。このずれは一回限りであり、その後のバックアップはスケジュール時間どおりに実行されます。

設定

ストラクチャー設定のバックアップ/設定ページではバックアップやログファイルの有効化/無効化、および保存先を設定できます。これらのパラメーターは、4D や 4D Server で開かれる各アプリケーションごとに設定されます。

The screenshot shows the 'myProject - Structure Settings' interface. The top navigation bar includes tabs for General, Interface, Compiler, Database, Moving, Backup (selected), Client-server, Web, SQL, PHP, Security, and Compatibility. Below this is a sub-navigation bar with Scheduler, Configuration, and Backup & Restore (selected). The main content area is divided into sections: Content (with checkboxes for Data, Structure, and User Structure), Attachments (listing files like ./Data/data.4DIndx, ./Data/data.4DSyncData, ./Data/data.4DSyncHeader), and a section for Backup File Destination Folder (set to "myProject" in volume "D:"). Log Management is also present with a checked checkbox for Use Log.

4D Server: これらのパラメーターは 4D Server マシン上でのみ設定できます。

内容

このエリアでは、次回のバックアップ時にコピー対象とするファイルやフォルダーを指定します。

- データ: アプリケーションのデータファイル。このオプションが選択されている場合、次のものがデータとともにバックアップされます:
 - データベースのカレントログファイル (あれば)
 - データファイルの隣に置かれた Settings フォルダー (あれば)。これは データファイル用のユーザー設定を格納しています。
- ストラクチャー: アプリケーションの Project フォルダーとファイル。プロジェクトがコンパイルされている場合には、このオプションは .4dz ファイルをバックアップします。このオプションがチェックされていると、Project フォルダーと同階層に置かれた Settings フォルダーが自動でバックアップされます。これは、ユーザー設定を格納しています。
- ユーザーストラクチャー(バイナリデータベースのみ): 廃止予定
- 添付: このエリアでは、アプリケーションと同時にバックアップの対象とするファイルやフォルダーを指定します。ここではどのようなタイプのファイル (ドキュメントやプラグイン、テンプレート、ラベル、レポート、ピクチャーなど) でも指定できます。個々のファイル、または丸ごとバックアップするフォルダーを個々に設定できます。添付エリアには、設定されたファイルのパスが表示されます。
 - 削除: 選択したファイルを添付エリアから取り除きます。
 - フォルダー追加...: バックアップに追加するフォルダーを選択するダイアログボックスを表示します。復元の場合、フォルダーがその内容物とともに復元されます。アプリケーションファイルを含むフォルダーを除き、すべてのフォルダーやマシンに接続されたボリュームを選択できます。
 - ファイル追加...: バックアップに追加するファイルを選択するダイアログボックスを表示します。

バックアップファイル保存先

このエリアではバックアップファイルの格納場所を確認したり、変更したりできます。

エリアをクリックすると、ファイルの場所がポップアップで表示されます。

バックアップファイルの格納場所を変更するには、 [...] ボタンをクリックします。選択ダイアログが表示され、バックアップファイルを配置するフォルダーやディスクを選択できます。"使用状況" と "空き容量" エリアは、選択したフォルダーが存在するディスクの状態を自動で表示します。

ログ管理

ログを使用 オプションが選択されていると、アプリケーションはログファイルを使用します。ログファイルの場所はオプションの下に表示されます。このオプションが選択されている場合、ログファイルなしでアプリケーションを開くことはできません。

デフォルトでは、4D で作成されたすべてのプロジェクトでログファイルが使用されます（環境設定 の 一般ページ 内でチェックされている ログを使用 オプションです）。ログファイルには *data.journal* のように名前が付けられ、Data フォルダー内に置かれます。

新しいログファイルを有効にするには、その前にアプリケーションのデータをバックアップしなければなりません。このオプションをチェックすると、バックアップが必要である旨の警告メッセージが表示されます：ログファイルの作成は延期され、実際には次のバックアップの後にログファイルが作成されます。

バックアップ & 復旧

バックアップ & 復旧の設定は必要に応じて変更します。デフォルトの設定は、標準的なバックアップ動作をおこないます。

myProject - Structure Settings

General Interface Compiler Database Moving Backup Client-server Web SQL PHP Security Compatibility

Scheduler Configuration Backup & Restore

General settings

Keep only the last backup files

Backup only if the data file has been modified

Delete oldest backup file

If backup fails: Retry at the next scheduled date and time
 Retry after Seconds

Cancel the operation after attempts

Archive

Segment Size (Mb):

Compression Rate:

Interlacing Rate:

Redundancy Rate:

Automatic Restore

Restore last backup if database is damaged

Integrate last log if database is incomplete

一般設定

- 最新のバックアップのみ保存 X バックアップファイル：このパラメーターを有効にすると、指定された数の最新バックアップファイルだけが保持され、古いバックアップファイルは削除されます。この機能は以下のように動作します：バックアップ処理が完了したら、アーカイブが作成されたと同じ場所、同じ名前のもっとも古いアーカイブを削除します。ディスクスペースを確保するため、バックアップ前に削除するよう、削除のタイミングを変更することもできます。たとえば、3世代のファイルを保持するよう設定している場合、最初の 3回のバックアップで MyBase-0001、MyBase-0002、MyBase-

0003 が作成され、4回目のバックアップで MyBase-0004 が作成されたのちに MyBase-0001 が削除されます。この設定はデフォルトで有効になっており、4D は 3世代のバックアップを保持します。このメカニズムを無効にするには、チェックボックスの選択を外します。

このパラメーターは、アプリケーションおよびログファイル両方のバックアップに影響します。

- データファイルが更新された場合のみバックアップを行う：このオプションが選択された場合、前回のバックアップ以降にデータが追加・変更・削除された場合のみ、4D は定期的なバックアップを開始します。そうでない場合、定期的なバックアップはキャンセルされ、次回のスケジュールまで延期されます。エラーは生成されませんが、バックアップジャーナルにはバックアップが延期された旨記録されます。このオプションを使用すれば、主に参照目的で使用されているアプリケーションのバックアップに消費されるマシン時間を節約できます。ストラクチャーや添付ファイルに対して変更がおこなわれていても、データファイルの更新としては扱われない旨注意してください。

このパラメーターは、アプリケーションおよびログファイル両方のバックアップに影響します。

- 最も古いバックアップファイルを削除：このオプションは "最新のバックアップのみ保存 X バックアップファイル" が有効になっている場合のみ使用されます。このオプションを使用して、最も古いバックアップファイルを削除するタイミングを設定します。選択肢は バックアップ前、あるいは バックアップ後 です。このオプションが機能するには、バックアップファイルが名称変更されたり、移動されたりしてはなりません。
- バックアップ失敗時：このオプションを使用して、バックアップ失敗時の処理を設定できます。バックアップが実行できなかった場合、4D では再試行することが可能です。
 - 次の予定された日付と時刻に再試行する：このオプションは、定期的な自動バックアップを設定されている場合にのみ意味があります。失敗したバックアップはキャンセルされます。エラーが生成されます。
 - 指定時間経過後に再試行：このオプションが選択されていると、設定された待ち時間経過後にバックアップを再試行します。このメカニズムを使用すると、バックアップをブロックするような特定の状況に対応することが可能となります。秒、分、あるいは時間単位で待ち時間を設定できます。次のバックアップ試行にも失敗するとエラーが生成され、ステータスエリアに失敗状況が表示され、バックアップジャーナルにも記録されます。
 - 操作をキャンセル X 試行後：このパラメーターを使用して、バックアップ試行の失敗最大数を設定できます。この最大数に達してもバックアップが正しく実行できなかった場合、バックアップはキャンセルされ、エラー 1401 ("バックアップ試行の最大数に達しました。自動バックアップは無効になります") が生成されます。この場合、データベースを再起動するか、手動バックアップが成功するまで自動バックアップはおこなわれません。このパラメーターは、人による介入が必要となるような問題があり、バックアップ試行が自動的に繰り返されることにより全体的なパフォーマンスに影響するようなケースで使用できます。デフォルトでこのオプションは選択されていません。

定期的なバックアップが実行される予定時刻にアプリケーションが起動されていなかった場合、4D はバックアップが失敗したものとして扱います。

アーカイブ

これらのオプションはメインのバックアップファイルとログバックアップファイルに適用されます。

- セグメントサイズ (MB)：4D ではアーカイブをセグメントに分割できます。この振る舞いにより、たとえばバックアップファイルを複数の異なるディスク (DVDやUSBデバイス等) に格納できます。復元時、4D はセグメントを自動的に統合します。各セグメントには MyApplication[xxxx-yyyy].4BK といった名称がつけられます (xxxx はバックアップ番号、yyyy はセグメント番号)。たとえば、MyApplication のバックアップが 3つのセグメントに分割されると、次のような名前になります: MyApplication[0006-0001].4BK, MyApplication[0006-0002].4BK, MyApplication[0006-0003].4BK セグメントサイズ はコンボボックスであり、各セグメントのサイズを MB単位で設定できます。メニューから定義済み値を選択するか、0~2048 の値を入力できます。0 を指定するとセグメント化はされません (なしを指定したのと同じ)。
- 圧縮率：デフォルトで 4D はバックアップファイルを圧縮し、ディスクスペースを節約します。しかし大量のデータがある場合、ファイルの圧縮処理はバックアップにかかる時間を長くします。圧縮率 オプションを使用してファイルの圧縮モードを調整できます：
 - なし：ファイルの圧縮はおこなわれません。バックアップは早くおこなわれますが、ファイルサイズは大きくなります。
 - 速度 (デフォルト)：このオプションはバックアップの速度とアーカイブサイズのバランスが考慮されたものです。
 - 圧縮率：アーカイブに最大の圧縮率が適用されます。アーカイブファイルはディスク上で最小のサイズとなります、バックアップの速度は低下します。
- インターレース率と冗長率：4D は最適化 (インターレース) とセキュリティ (冗長) メカニズムに基づく特定のアルゴリズムを使用してアーカイブを生成します。これらのメカニズムを必要に応じて設定できます。これらのオプションのメニューには低・中・高・なし (デフォルト) の選択肢があります。
 - インターレース率：インターレースとはデータを連続しない領域に書き込むことにより、セクター損傷の際のリスクを低減させるものです。率を上げることでリスクがより低減されますが、データの処理により多くのメモリが必要となります。
 - 冗長率：冗長は同じ情報を複数回繰り返すことで、ファイル中のデータを保護するものです。冗長率を高くするとよりファイルが保護されます。しかし書き込みは遅くなり、ファイルサイズも増大します。

自動復元

- データベースが壊れていたら、最新のバックアップから復元する：このオプションが選択されていると、ファイル破損などの異常が検知された場合、4D は起動時にアプリケーションの有効な最新のバックアップからのデータの復旧を自動で開始します。ユーザーによる介入は必要ありませんが、処理はバックアップジャーナルに記録されます。
- データベースが完全でない場合、最新のログを統合する：このオプションがチェックされると、プログラムはアプリケーションを開く際または復旧時に、自動でログファイルを統合します。
 - アプリケーションを開く際に、データファイルに保存されていない処理がログファイル中に見つかった場合、4D は自動でカレントログファイルを統合します。このようなケースは、ディスクに書き込まれていないデータがまだキャッシュ中に存在する状態で、電力の切断が起きた場合に発生します。
 - アプリケーションの復元時、カレントログファイルや、バックアップファイルと同じ番号を持つログバックアップファイルが同じフォルダーに存在する場合、4D はその内容を検証します。そしてデータファイルに書き込まれていない処理が見つかれば、自動で統合処理がおこなわれます。

ユーザーにダイアログボックスが提示されることはありません。処理は完全に自動です。処理はバックアップジャーナルに記録されます。

自動復元の場合、復元されるのは次の要素に限られます：

- .4DD ファイル
- .4DIndx ファイル
- .4DSyncData ファイル
- .4DSyncHeader ファイル
- External Data フォルダー

添付ファイルやプロジェクトファイルを取得したい場合、[手動の復元](#) をおこなう必要があります。

ログファイル (.journal)

継続的に使用されるアプリケーションでは、変更・追加・削除の操作は常に記録されます。定期的にデータをバックアップすることは重要ですが、バックアップだけでは（予期しない障害の場合に）、前回のバックアップ以降に入力されたデータを回復することができません。この必要性に対応するため、4Dは専用ツールであるログファイルを提供しています。このファイルを使用すると、データのセキュリティが常に保証されます。

また、4Dは常にメモリー上のデータキャッシュを使用して作業をおこなっています。アプリケーションのデータへの変更はすべて、ハードディスクへ書き込む前に、キャッシュへ一時的に保存されます。これにより、アプリケーションの処理速度が向上します。実際、メモリーへのアクセスは、ハードディスクへのアクセスよりも高速です。キャッシュに保存したデータをディスクへ書き込む前に障害が発生した場合は、カレントログファイルを組み込んでアプリケーションを完全に復旧しなくてはなりません。

さらに4Dには、カレントログファイルの内容を解析する機能が組み込まれており、これによってアプリケーションのデータ上で実行されたすべての処理をさかのぼることができます。これらの機能はMSCにて提供されています（[ログ解析](#)ページおよび[ロールバック](#)ページ参照）。

ログファイルについて

4Dが生成するログファイルには、アプリケーション上でおこなわれた操作がすべて順次記録されています。デフォルトでは、すべてのテーブルのデータログが取られています（つまりログファイルに含まれています）。しかし、ログファイルに含めるプロパティを選択解除することによって特定のテーブルをログに含めないようにすることもできます。

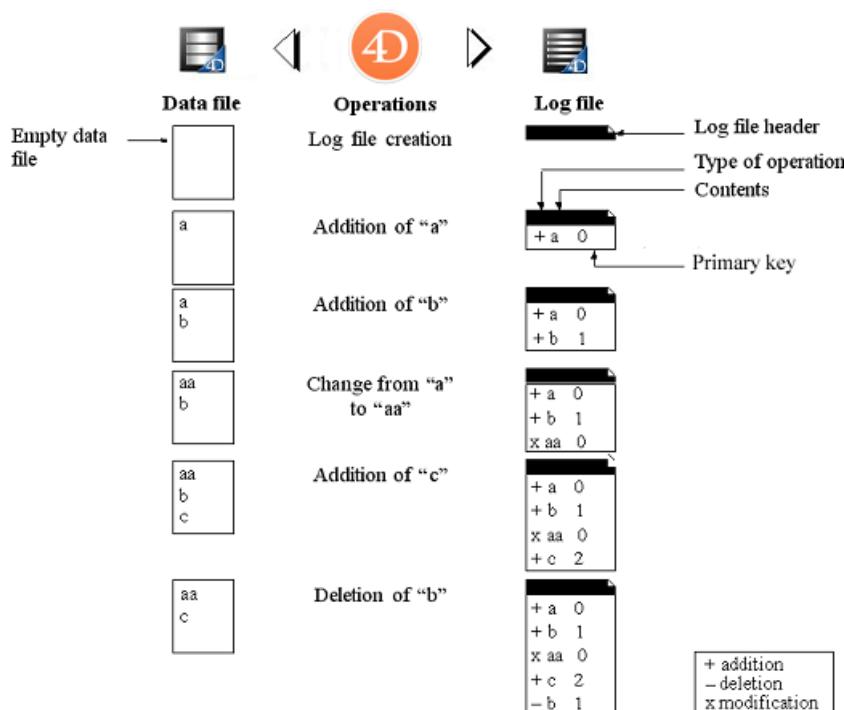
したがって、ユーザーが実行した各操作により、2つのアクションが同時に起こります。1つは、データファイルに対するアクション（命令を通常どおりに実行）、もう1つはログファイルに対するアクション（処理の説明を記録）です。ログファイルは個別に作成され、ユーザーの作業を妨げたり作業速度を低下させることはありません。1つのアプリケーションでは、一度に1つのログファイルだけを扱うことができます。ログファイルには、次の操作が記録されます：

- データファイルの開閉
- プロセス（コンテキスト）の開閉
- レコードまたはBLOBの追加
- レコードの変更
- レコードの削除
- トランザクションの作成や終了

これらのアクションについての詳細は、MSCの[ログ解析](#)ページを参照ください。

ログファイルは4Dにより管理されます。ログファイルは、データファイルに影響を与えるすべての操作を区別なく盛り込み、ユーザーがおこなった操作や4Dメソッド、SQLエンジン、プラグイン、Webブラウザーやモバイルアプリなどによる処理など、あらゆる操作を記録します。

ログファイルの機能をまとめた図を次に示します：



カレントログファイルはカレントデータファイルと一緒に自動保存されます。このメカニズムには、2つの際立った利点があります：

- ログファイルが保存されるディスクの容量が一杯にならないようにします。バックアップを実行しない場合、ログファイルは使用するにつれて徐々に大きくなり、いずれはディスクの空き容量をすべて使い果たしてしまいます。データファイルをバックアップするたびに、4D や 4D Server はカレントログファイルをクローズし、その後に空ファイルを新たに開くため、ディスクフルになる危険を避けることができます。その後、古いログファイルはアーカイブに保存され、バックアップのセット（世代）を管理するメカニズムに従って最終的には破棄されます。
- 後からアプリケーションの解析や修復をおこなえるように、各バックアップに対応するログファイルを保管します。ログファイルの統合は、それが対応するアプリケーションからのみ実行できます。バックアップに正しくログファイルを統合するため、バックアップとアーカイブされたログファイルは一緒に保管することが重要です。

ログファイルの作成

デフォルトでは、4D で作成されたすべてのアプリケーションでログファイルが使用されます（環境設定の「一般」ページ内でチェックされているオプションです）。ログファイルには *data.journal* のように名前が付けられ、Data フォルダー内に置かれます。

アプリケーションでログファイルが使用されているかどうかは、いつでも調べることができます。これには、ストラクチャー設定の「バックアップ/設定」ページで「ログを使用」オプションが選択されているか確認します。このオプションの選択が解除されていた場合、またはログファイルなしでアプリケーションを使用している場合で、ログファイルを用いたバックアップ方法を導入するには、ログファイルを作成する必要があります。

ログファイルを作成するには、次の手順に従ってください：

1. ストラクチャー設定の「バックアップ/設定」ページで、「ログを使用」オプションを選択します。標準の「ファイルを開く/新規作成」ダイアログボックスが表示されます。ログファイルにはデフォルトで *data.journal* という名前が付けられます。
2. デフォルトの名前を使用するか、またはその名前を変更し、次にファイルの保管場所を選択します。2つ以上のハードドライブが存在する場合は、アプリケーションプロジェクトが保管されているディスク以外の場所にログファイルを保存することをお勧めします。これにより、アプリケーションが保管されているハードドライブが破損した場合でも、ログファイルを呼び出すことができます。
3. 「保存」をクリックします。開いたログファイルのアクセスパスと名前がダイアログボックスの「ログを使用」エリアに表示されます。このエリアをクリックすると、ポップアップメニューが表示され、ディスク上のフォルダーを確認できます。
4. ストラクチャー設定ダイアログボックスを確定します。

ログファイルを作成するには、データが次の条件のいずれかを満たしていないなりません：

- データファイルが空である。
- バックアップを実行した直後であり、データへの変更がまだおこなわれていない。

いずれの条件も満たしていない場合は、バックアップを実行する必要がある旨を知らせる警告ダイアログボックスが表示されます。OK をクリックするとバックアップが開始され、その後にログファイルが作成されます。キャンセルをクリックした場合には、ログファイル作成の要求は保存され、次回アプリケーションをバックアップする時までログファイルの作成は延期されます。このような安全対策が不可欠な理由は、障害の発生後にアプリケーションを復元するために、ログファイルへ記録された処理を統合するアプリケーションのコピーが必要となるからです。

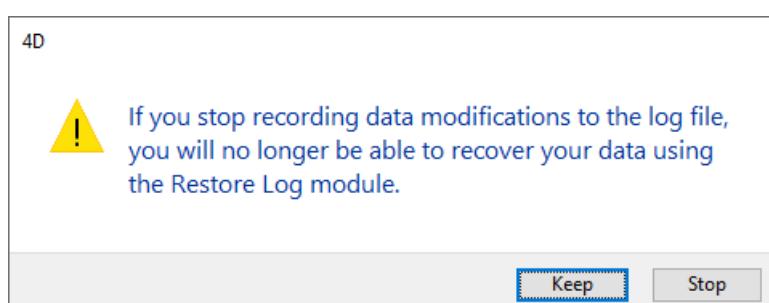
これ以外に何もおこなわなくても、データ上で実行されたすべての処理がこのファイルに記録され、その後アプリケーションを開いたときにこのファイルが使用されます。

新規データファイルを作成したら、別のログファイルを作成しなくてはなりません。また、ログファイルに関連付けられていない（あるいはログファイルが見つからない）別のデータファイルを開いた場合、他のログファイルを設定するか、作成しなくてはなりません。

ログファイルを中止する

カレントログファイルへの操作記録を中止したい場合は、ストラクチャー設定の「バックアップ/設定」ページの「ログを使用」オプションを選択解除します。

すると、4D は警告メッセージを表示して、この動作によりログファイルによるセキュリティが利用できなくなることを知らせます：



停止 をクリックすると、カレントログファイルが即座にクローズされます（この後にストラクチャー設定ダイアログボックスを確定する必要はありません）。

カレントログファイルが大きすぎるため、それをクローズしたい場合は、データファイルのバックアップを実行してください。これにより、ログファイルのバックアップが作成されます。

4D Server: `New log file` コマンドはカレントログファイルを自動的に閉じて、新しいログファイルを開始します。実行中に何らかの理由でログファイルが利用不能になった場合、エラー 1274 が生成され、4D Server は一切のデータ書き込みを許可しなくなります。再びログファイルが利用可能になつたらフルバックアップを実行しなければなりません。

復元

問題が発生したときは、一連のアプリケーションファイル全体を復元することができます。主に 2つのカテゴリの問題が発生する可能性があります：

- アプリケーションが使用中に予期せず終了された。この問題は電力の切断、システムのエラー等により発生する可能性があります。この場合、問題が発生した瞬間のデータキャッシュの状態により、アプリケーションの復旧には異なる手順が必要となります：
 - キャッシュが空の場合、アプリケーションを問題なく開くことができます。アプリケーションに対しておこなわれた変更はデータファイルに記録されています。この場合には、特別な手順は必要ありません。
 - キャッシュに未保存の処理が含まれている場合、データファイルは損傷していませんが、カレントのログファイルを統合する必要があります。
 - キャッシュの内容をデータファイルに書き込み中だった場合、データファイルはおそらく損傷しています。最新のバックアップから復元をおこない、カレントのログファイルを統合する必要があります。
- アプリケーションファイルを失った。この問題はアプリケーションが配置されたディスク上のセクターが読み書き不能になった、あるいはウイルス、操作ミス等により発生します。最新のバックアップから復元をおこない、カレントのログファイルを統合する必要があります。問題発生後にアプリケーションが損傷しているかどうかを見分けるには、4D でアプリケーションを起動します。4Dは自己検証をおこない、必要な復元処理手順を示します。自動モードの場合、この処理はユーザーのアクションなしで直接実行されます。定期的なバックアップがおこなわれていれば、4D の復元ツールを使用して（ほとんどの場合）問題が発生する直前の状態までアプリケーションを復旧することができます。

問題発生後に、自動で 4Dのアプリケーション復旧処理を起動することができます。このメカニズムは、ストラクチャ設定の バックアップ/バックアップ&復旧 ページで利用できるオプションを使用して管理します。詳細は [自動復元](#) を参照してください。

問題が、データに対しておこなわれた不適切な処理の結果引き起こされた場合（たとえば誤ってレコードを削除した等）、ログファイルの "ロールバック" 機能を使用してデータファイルを復旧できます。この機能は MSC の [ロールバック](#) ページから利用できます。

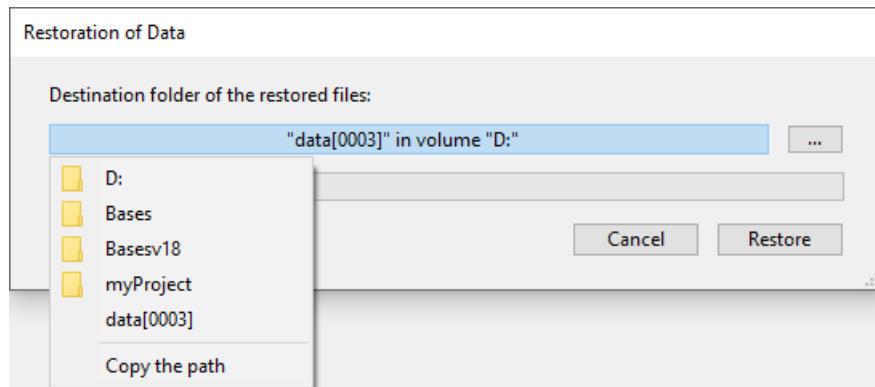
手動でバックアップから復元する (標準ダイアログ)

バックアップモジュールを使用して生成されたアーカイブの内容を、手動で復元することができます。手動による復元は、たとえばアーカイブ全体（ストラクチャーファイルや添付されたファイル）を再生成したい場合や、アーカイブの内容を見たい場合などに必要となります。手動復元の際に、カレントログファイルを統合することもできます。

バックアップの手動復元は、標準のファイルを開くダイアログボックス、あるいは Maintenance and Security Center (MSC) の [復元](#) ページからおこなうことができます。MSC を使用した復元では詳細なオプション設定をおこなったり、アーカイブの内容をプレビューしたりすることができます。他方、開かれているアプリケーションに関連したアーカイブのみを復元できます。

標準ダイアログボックスを使用してアプリケーションを手動復元するには：

1. 4Dアプリケーションを開始し、ファイル メニューから 復元... を選択します。アプリケーションプロジェクトが開かれている必要はありません。または 4Dメソッドから RESTORE コマンドを実行します。標準のファイルを開くダイアログボックスが表示されます。
2. 復元するバックアップファイル (.4bk) またはログバックアップファイル (.4bl) を選択し、開くをクリックします。復元したファイルを配置する場所を指定するために、以下のダイアログボックスが表示されます：デフォルトで 4Dはアーカイブと同階層にアーカイブ名と同じ名前（拡張子なし）のフォルダを作成し、ファイルを復元します。場所が表示されているエリアをクリックして、パスを確認することができます：



[...] ボタンをクリックして異なる場所を指定することもできます。

3. 復元 ボタンをクリックします。4D は指定されたすべてのバックアップファイルを展開します。カレントログファイル、または、バックアップファイルと同じ番号を持つログバックアップファイルが同じフォルダーに存在する場合、4D はその内容を検証します。データファイル中に無い処理がログファイルに含まれる場合は、4Dはそれをアーカイブに含めます。

れていれば、その処理を統合するかどうか 4D が尋ねてきます。データベースが完全でない場合、最新のログを統合するオプションが選択されている場合、統合処理は自動でおこなわれます（[自動復元 参照](#)）。

4. (任意) OK をクリックして、復元したアプリケーションにログファイルを統合します。復元と統合が正しく実行されると、4D は処理が成功したこと通知するダイアログを表示します。
5. OK をクリックします。

保存先フォルダーが表示されます。バックアップ時のファイルの位置にかかわらず、4D はすべてのバックアップファイルをこのフォルダーに配置します。これにより、ファイルを探す手間が省けます。

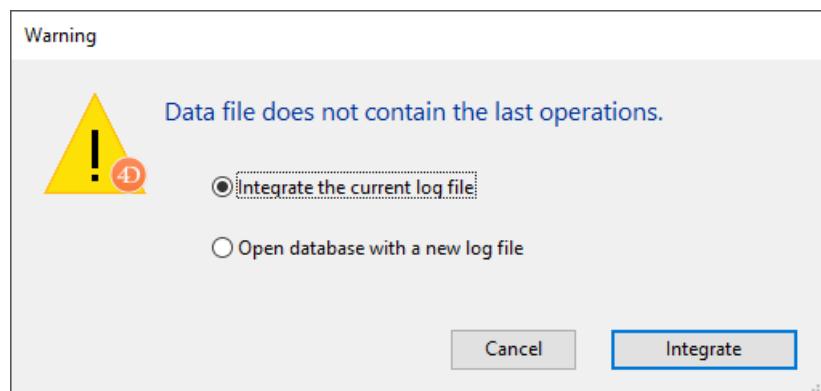
データファイルの関連要素（ファイルや **Settings** フォルダー）は保存先フォルダー内の **Data** サブフォルダー内に自動的に復元されます。

手動でバックアップから復元する (MSC)

Maintenance and Security Center (MSC) の [復元](#)ページから、カレントアプリケーションのアーカイブを手動で復元できます。

手動でログを統合する

MSC の復元ページでログファイルの自動統合を選択していない場合（[複数のログファイルを連続して統合する 参照](#)）、データファイルに保存されていない処理がログファイル中に見つかると、4D はアプリケーションを開く際に警告ダイアログボックスを表示します。



このメカニズムを機能させるために、4D はカレントの場所にあるログファイルにアクセスできなければなりません。

カレントログファイルを統合するかしないかを選択することができます。カレントログファイルを統合しないことにより、データ中に作成されたエラーを再生成しないようにすることもできます。

概要

4D のローカルモード、リモートモード、および 4D Server には Webサーバーエンジン (HTTPサーバー) があります。この Webサーバーエンジンを使用して、4Dデータベースを最大限に活用できる強力な Webアプリケーションを設計・公開することができます。

簡単なモニタリング

Webアプリケーションの公開は、いつでも開始または停止することができます。メニュー命令を選択、またはランゲージ命令を実行するだけで、操作できます。

4D Webサーバーの監視も簡単で、4D Server の管理ウィンドウや [専用URL](#) を使っておこなうことができます。

すぐに使えます

4D Webサーバーは、デフォルトのルートフォルダーとデフォルトのホームページを自動作成するため、すぐに利用できます。

セキュリティ

データセキュリティは、4D Webサーバーの実装のすべての段階に存在します。セキュリティレベルは調整可能で、デフォルト設定では通常、もっとも安全なオプションが選択されます。4D Webサーバーのセキュリティは、以下の要素に基づいています：

- [TLSプロトコル \(HTTPS\)](#) の拡張サポート。
- 認証：ビルトインの設定および、フォールバックデータベースメソッド (Webサーバー用の [On Web Authentication](#)、RESTサーバー用の [On REST Authentication](#)) に基づく柔軟でカスタマイズ可能な [認証機能](#)
- 公開するコンテンツの管理：明示的に公開した要素のみが、Web や RESTリクエストで直接利用できます。次のものについて、宣言する必要があります：
 - HTTPリクエストで公開する [プロジェクトメソッド](#)
 - RESTリクエストで公開する [ORDAのデータモデルクラス関数](#)
 - RESTリクエストに公開しない [テーブルやフィールド](#) テーブルやフィールド
- [デフォルトHTMLルート](#) フォルダーを定義することによる サンドボックス化
- サーバーによるリソース使用の管理 (例: [最大同時Webプロセスオプション](#))

4Dのセキュリティ機能の概要については、[4D Security guide](#) をご覧ください。

ユーザーセッション

4D Webサーバーには cookie を使用する、完全に自動化された [Webセッション](#) (ユーザーセッション) 管理機能があります。

RESTリクエストへのゲートウェイ

4D Webサーバーにより、4Dアプリケーションに保存されているデータに RESTリクエストを通じてアクセスすることが可能になります。RESTリクエストによって、データの追加・読み取り・編集・並べ替え・検索など、あらゆるデータベース操作に直接アクセスできます。

RESTリクエストの詳細については、[RESTサーバー](#) のセクションを参照ください。

拡張設定

4D Webサーバーの構成は、アプリケーションレベルの包括的な設定によって定義されます。この設定は、[webServer](#) オブジェクトのプロパティまたは [WEB SET OPTION](#) コマンドを使用して、セッション毎にカスタマイズすることもできます。

テンプレートとURL

4D Webサーバーは、テンプレートページおよび専用のURLを介して、4Dアプリケーションに保存されているデータへのアクセスを提供します。

- テンプレートページには、ブラウザに送信される際に Webサーバーの処理を開始する [特別なタグ](#) が含まれています。
- [専用のURL](#) は、任意のアクションを実行するために 4D を呼び出すもので、ユーザーが HTMLフォームを POST したときに処理を開始するフォームアクションとしても使用できます。

専用のデータベースメソッド

`On Web Authentication`、`On Web Connection`、および `On REST Authentication` データベースメソッドは、Webサーバーにおいてリクエストのエントリーポイントであり、あらゆるタイプのリクエストを評価・ルーティングするのに使用できます。

設定

4D Webサーバーの設定には、セキュリティパラメーター、リスニングポート、デフォルトのパス、およびサーバーの機能を網羅するさまざまなオプションが含まれます。4D ではすべての設定にデフォルト値を用意しています。

設定をおこなう場所

4D Webサーバーの設定には、スコープやサーバーに応じた様々な方法があります：

設定場所	スコープ	使用する Webサーバー
webServer オブジェクト	一時的 (カレントセッション)	コンポーネントWebサーバーを含む、あらゆる Webサーバー
WEB SET OPTION または WEB XXX コマンド	一時的 (カレントセッション)	メインサーバー
ストラクチャー設定 ダイアログボックス (Web ページ)	永続的 (全セッション、ディスク上に保存)	メインサーバー

設定できる場所が限られる設定も一部存在します。

キャッシュ

設定できる場所	名称	コメント
設定ダイアログボックス	オプション (I) ページ / 4D Webキャッシュを使用する	
設定ダイアログボックス	オプション (I) ページ / ページキャッシュサイズ	

Webページキャッシュの有効化と設定をおこないます。

4D Webサーバーにはキャッシュがあり、スタティックページ、GIF、JPEG (<512 kb)、そしてスタイルシート (.css ファイル) などがリクエストされると、メモリにロードされます。キャッシュの利用は、スタティックページの送信時に Webサーバーのパフォーマンスを大幅に向上します。キャッシュはすべての Webプロセスで共有されます。キャッシュが有効化されている場合、4D Webサーバーは、ブラウザーからリクエストされた静的ページをまずキャッシュ内で探します。ページが見つかれば、それを送信します。見つからない場合、4D はディスクからページを読み込み、キャッシュに格納します。

キャッシュのサイズは、ページキャッシュサイズ エリアで変更できます。設定する値は、スタティックページのサイズや数、およびホストマシンで利用可能なりソースによります。

Webデータベースを利用する間、WEB GET STATISTICS コマンドを使用してキャッシュのパフォーマンスを検証できます。たとえば、キャッシュ利用率が 100% に近い場合、キャッシュに割り当てたメモリ量を増やすことを考慮します。[/4DSTATS] と [/4DHMLSTATS] の URL も、キャッシュの状態を知るのに使用できます。

証明書フォルダー

設定できる場所	名称	コメント
webServer オブジェクト	certificateFolder	テキストプロパティ (start() 関数の settings パラメーターと使用する場合は、4D.Folder オブジェクトも使用可能)

Webサーバー用の TLS証明書ファイルが置かれているフォルダーです。

4D または 4D Server のデフォルトでは、これらのファイルは Project フォルダー の隣に配置する必要があります。

4D をリモートモードで使用する場合、これらのファイルは、リモートマシン上のデータベースのローカルリソースフォルダーに配置されている必要があります

(`Get 4D folder` コマンドの `4D Client Database Folder` の項を参照ください)。これらのファイルをリモートマシンに手動でコピーする必要があります。

TLS 証明書ファイルは、`key.pem` (秘密の暗号鍵を含むドキュメント) と `cert.pem` (証明書を含むドキュメント) です。

文字コード

設定できる場所	名称	コメント
webServer オブジェクト	<code>characterSet</code>	MIBEnum 整数、または名称の文字列
<code>WEB SET OPTION</code>	<code>Web character set</code>	MIBEnum 整数、または名称の文字列
設定ダイアログボックス	オプション (II) ページ / 文字コード	ポップアップメニュー

4D Webサーバーが使用する文字セットを定義します。デフォルト値は OS の言語に依存します。

この設定は、クリックレポートを HTMLフォーマットで書き出す際にも使用されます。

暗号リスト

設定できる場所	名称	コメント
webServer オブジェクト	<code>cipherSuite</code>	テキスト

セキュアプロトコルに使用される暗号リストです。Webサーバーが実装する暗号アルゴリズムの優先順位を設定します。コロン区切りの文字列として設定できます (例: "ECDHE-RSA-AES128-...")。詳細は Open SSL サイトの [ciphers ページ](#) を参照ください。

4D が使用するデフォルトの暗号リストは、`SET DATABASE PARAMETER` コマンドを使用してセッションごとに変更することができます。この場合、変更は 4D アプリケーション全体に適用されます (Webサーバー・SQLサーバー・クライアント/サーバー接続、HTTPクライアント、セキュアプロトコルを使用するすべての 4Dコマンドを含む)。

CORS設定

設定できる場所	名称	コメント
webServer オブジェクト	CORSSettings	オブジェクトのコレクション (CORSサービスで許可されたホストとメソッドの一覧)
<code>WEB SET OPTION</code>	<code>Web CORS settings</code>	オブジェクトのコレクション (CORSサービスで許可されたホストとメソッドの一覧)
設定ダイアログボックス	オプション (II) ページ / ドメイン名 および 許可された HTTPメソッド	新しいドメインとメソッドを許可するには [+] ボタンをクリックして追加します。

CORSサービスで許可されたホストとメソッドの一覧。

ドメイン名 (hostプロパティ)

CORS を介したサーバーへのデータリクエスト送信が許可されている外部ページのドメイン名または IPアドレス。複数のドメインを追加してホワイトリストを作成することができます。複数のシンタックスがサポートされています:

- 192.168.5.17:8081
- 192.168.5.17
- 192.168.*
- 192.168.*:8081
- <http://192.168.5.17:8081>

- http://*.myDomain.com
- <http://myProject.myDomain.com>
- *.myDomain.com
- myProject.myDomain.com
- *

許可された HTTPメソッド (methodsプロパティ)

対応する CORSホストに対して許可する HTTPメソッド。以下の HTTPメソッドがサポートされます:

- GET
- HEAD
- POST
- PUT
- DELETE
- OPTIONS
- TRACE
- PATCH

メソッド名はセミコロン区切りで指定します(例: "post;get")。methods が空、null、あるいは undefined の場合、すべてのメソッドが許可されます。

参照

[CORSを有効化](#)

デバッグログ

設定できる場所	名称	コメント
webServer オブジェクト	debugLog	number
WEB SET OPTION	Web debug log	number

Webサーバーの HTTPリクエストログファイル (アプリケーションの "Logs" フォルダーに格納されている [HTTPDebugLog_nn.txt](#) (nn はファイル番号)) の状態を指定します。このログファイルは、Webサーバーに関連する問題をデバッグするのに便利です。ログには、各リクエスト・レスポンスが raw モードで記録されます。ヘッダーを含むリクエスト全体が記録され、オプションでボディ部分も記録することができます。

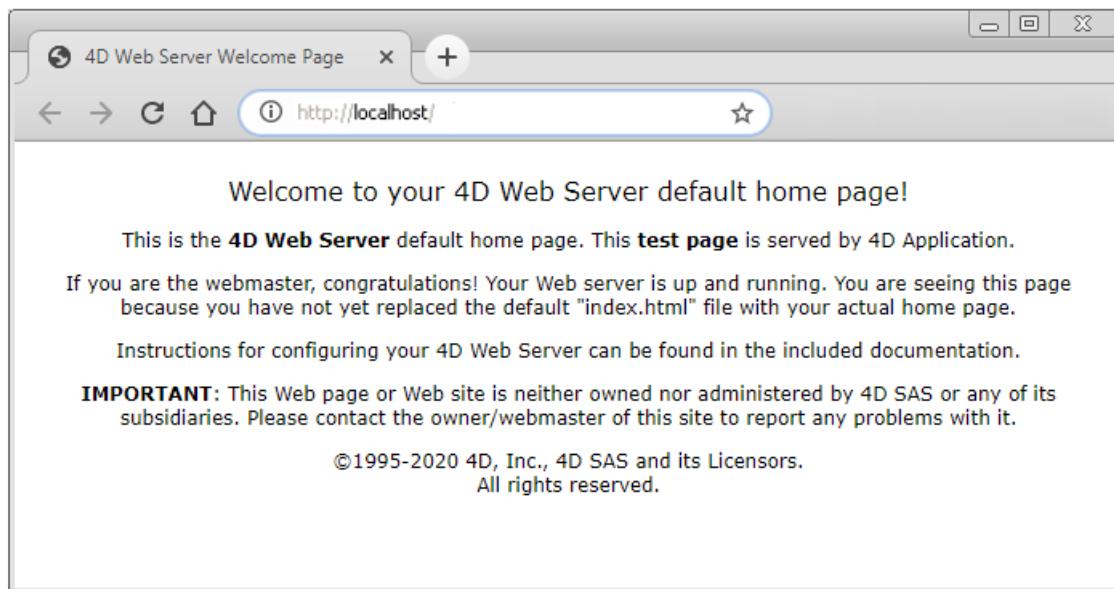
値	定数	説明
0	wdl disable	Web HTTP debug log は無効化されています
1	wdl enable without body	Web HTTP debug log 有効、リクエスト本文なし (本文サイズあり)
3	wdl enable with response body	Web HTTP debug log 有効、レスポンスの本文のみ
5	wdl enable with request body	Web HTTP debug log 有効、リクエストの本文のみ
7	wdl enable with all body parts	Web HTTP debug log 有効、リクエストおよびレスポンスの本文あり

デフォルトホームページ

設定できる場所	名称	コメント
webServer オブジェクト	defaultHomepage	テキスト
WEB SET HOME PAGE		Webプロセス毎に異なる設定が可能
設定ダイアログボックス	設定ページ / デフォルトホームページ	

Webサーバーのデフォルトホームページを指定します。このページはスタティックでもセミダイナミックでも可能です。

Webサーバーの初回起動時には、4Dはデフォルトで "index.html" という名前のホームページを作成し、HTMLルートフォルダーに置きます。この設定を変更しない場合、Webサーバーに接続するブラウザーには以下のようなページが表示されます：



デフォルトホームページを変更するには、パスを "デフォルトホームページ" エリアに入力します。

- パスは、[デフォルトHTMLルート](#)からの相対パスで設定しなければなりません。
- パスは POSIX シンタックスで表します（フォルダーはスラッシュ（"/"）で区切れます）。
- パスはスラッシュ（"/"）で始まつたり終わつたりしてはいけません。

たとえば、デフォルトHTMLルートフォルダー内の "Web" サブフォルダーにある "MyHome.htm" をデフォルトホームページにする場合、"Web/MyHome.htm" と入力します。

デフォルトホームページを指定しない場合、[On Web Connection](#) データベースメソッドが呼び出されます。この場合には、プロシージャでリクエストを処理するのは開発者の役割です。

CORSを有効化

設定できる場所	名称	コメント
webServer オブジェクト	CORSEnabled	ブール； CORSを有効化するには true (デフォルト値は false)
WEB SET OPTION	Web CORS enabled	0 (デフォルト値； 無効) または 1 (有効)
設定ダイアログボックス	オプション (II) ページ / CORSを有効化	デフォルトではチェックなし

4D Webサーバーは、クロスオリジンリソースシェアリング (CORS) を実装しており、これによって別ドメインにて提供されている特定の Webページが、REST などを使用した XHRコールを介してカレントWebアプリケーションのリソースにアクセスできるようになります。セキュリティ上の理由により、"ドメイン間" のリクエストはブラウザーレベルでデフォルトで禁止されています。有効化されている場合、ドメイン外 Webページからの XHRコール (RESTリクエストなど) をアプリケーションにおいて許可することができます (CORSドメインリストに許可されたアドレスのリストを定義する必要があります)。有効時に、許可されていないドメインやメソッドがサイト間リクエストを送信した場合、"403 - forbidden" エラーレスポンスによって拒否されます。

無効化されている場合 (デフォルト) には、CORS で送信されたサイト間リクエストはすべて無視されます。

CORSについての詳細は、Wikipedia の [Cross-origin resource sharing](#) ページを参照ください。

参照

[CORS設定](#)

HTTPを有効化

設定できる場所	名称	コメント
webServer オブジェクト	HTTPEnabled	boolean
WEB SET OPTION	Web HTTP enabled	
設定ダイアログボックス	設定ページ / HTTPを有効化	

安全でない接続を Webサーバーが受け入れるかどうかを示します。

HTTPSを有効にする

設定できる場所	名称	コメント
webServer オブジェクト	HTTPSEnabled	boolean
WEB SET OPTION	Web HTTPS enabled	
設定ダイアログボックス	設定ページ / HTTPSを有効にする	

Webサーバーがセキュアな接続を受け入れるか受け入れないかを指定します。このオプションは [TLSプロトコル](#) で説明しています。

HSTSを有効にする

設定できる場所	名称	コメント
webServer オブジェクト	HSTSEnabled	ブール; HSTSを有効化するには true (デフォルト値は false)
WEB SET OPTION	Web HSTS enabled	0 (デフォルト値; 無効) または 1 (有効)

HTTP Strict Transport Security (HSTS) の状態です。

[HTTPSを有効](#) にした場合、同時に [HTTPも有効](#) になっていると、ブラウザー上では HTTPS と HTTP を切り替えることができることに注意が必要です（たとえば、ブラウザーのアドレスバーで、ユーザーは "https" を "http" に置き換えることができます）。HTTPへのリダイレクトを禁止するには、[HTTPを無効](#) にすることもできますが、この場合、クライアントの HTTPリクエストに対してエラーメッセージが表示されてしまいます。

HSTS によって、4D Webサーバーはブラウザーに対し、セキュアな HTTPS接続のみを許可すると宣言できます。HSTS を有効にすると、4D Webサーバーはすべてのレスポンスヘッダーに HSTS 関連の情報を自動的に追加します。4D Webサーバーからの初回レスポンスを受け取った際にブラウザーは HSTS情報を記録し、以降の HTTPリクエストは自動的に HTTPSリクエストに変換されます。ブラウザー側でこの情報が保存される時間は HSTS max age 設定によって指定されます。

HSTS のためには、サーバー上で [HTTPS が有効](#) になっていないなりません。また、クライアントの初回接続を許可するために、[HTTP も有効](#) でなくてはなりません。

現在の接続モードは、[WEB Is secured connection](#) コマンドで取得できます。

HSTS Max Age

設定できる場所	名称	コメント
webServer オブジェクト	HSTSMaxAge	数値 (秒単位)
WEB SET OPTION	Web HSTS max age	数値 (秒単位)

新規クライアント接続ごとに HSTS がアクティブな最長時間 (秒単位) を指定します。この情報はクライアント側で指定された時間のあいだ保存されます。デフォルト値は 63072000 (2年)。

警告: HSTSを有効にすると指定された期間中、クライアントの接続はこのメカニズムを使用し続けます。アプリケーションをテストする際には、セキュアな接続とセキュアでない接続を必要に応じて切り替えることができるよう、短い期間を設定することが推奨されます。

HTTP圧縮レベル

設定できる場所	名称	コメント
webServer オブジェクト	HTTPCompressionLevel	
WEB SET OPTION	<code>Web HTTP compression level</code>	Web および Webサービスに適用されます

4D Webサーバーの HTTP圧縮通信 (クライアントリクエストまたはサーバーレスポンス) における圧縮レベル。この設定を使って、実行速度を優先するか (圧縮少)、それとも圧縮レベルを優先するか (速度減) を指定し、通信を最適化することができます。適切な値は、通信データのサイズとタイプによって異なります。

1 から 9 の値を指定します (1 が低圧縮、9 が高圧縮)。-1 を指定すると、圧縮速度と圧縮率の妥協点が得られます。デフォルトの圧縮レベルは 1 (低圧縮) です。

HTTP圧縮のしきい値

設定できる場所	名称	コメント
webServer オブジェクト	HTTPCompressionThreshold	
WEB SET OPTION	<code>Web HTTP compression threshold</code>	

最適化されたHTTP通信のフレームワークにおける、HTTP圧縮のしきい値 (バイト単位)。このサイズ未満のリクエストについては、通信が圧縮されません。この設定は、通信サイズが小さい場合、圧縮に処理時間が費やされるのを避けるのに有用です。

サイズを数値 (バイト単位) で指定します。デフォルトのしきい値は 1024 バイトに設定されています。

HTTP ポート

設定できる場所	名称	コメント
webServer オブジェクト	HTTPPort	number
WEB SET OPTION	<code>Web port ID</code>	
設定ダイアログボックス	設定ページ / HTTPポート	

HTTP接続を受け付ける IP (TCP) ポート番号。デフォルトで、4D は通常の Web HTTPポート (TCPポート) 番号である 80番を使用して Webアプリケーションを公開します。他の Webサービスによってこのポート番号が既に使用されている場合、4D が使用する HTTPポート番号を変更する必要があります。

macOS では、HTTPポートを変更することで、rootユーザーでなくても Webサーバーを開始することができるようになります ([macOS での Helperツール 参照](#))。

デフォルトでない HTTPポート番号を使用して公開された Webアプリケーションに接続するには、Webブラウザーで入力するアドレスにポート番号を含めなければなりません。アドレスの後にコロンに続けてポート番号を指定します。たとえば、HTTPポート番号 8080を使用する場合、"123.4.567.89:8080" のように書きます。

警告: デフォルトの TCPポート番号 (標準モードで 80、HTTPSモードで 443) 以外を指定する場合、同時に使用する他のサービスのデフォルトポート番号を使わないよう注意が必要です。たとえば、Webサーバーマシンで FTPプロトコルを使用する計画である場合、このプロトコルのデフォルトである TCPポート 20 と 21 を使用してはいけません。256 より下のポート番号は、well-known サービスに予約されています。また、256 から 1024 は UNIXプラットフォーム由来のサービスに予約されています。セキュリティのため、これらの数値よりも上、たとえば 2000 台や 3000台などを指定します。

0 を指定すると、4D はデフォルトの HTTPポート番号 80を使用します。

HTTP Trace

設定できる場所	名称	コメント
webServer オブジェクト	HTTPTrace	ブール; デフォルトは false
WEB SET OPTION	Web HTTP TRACE	数値; デフォルトは 0 (無効)

4D Webサーバーの HTTP TRACE メソッドを有効化します。セキュリティ上の理由により、4D Webサーバーはデフォルトで HTTP TRACE リクエストをエラー 405 で拒否します。必要に応じて有効化された場合、HTTP TRACE リクエストに対して Webサーバーは、リクエスト行、ヘッダー、および本文を返します。

HTTPS ポート

設定できる場所	名称	コメント
webServer オブジェクト	HTTPSPort	number

| WEB SET OPTION | Web HTTPS port ID ||

| Settings dialog box | Configuration page / HTTPS Port ||

TLS を介した HTTPS 接続を受け付ける IP ポート番号。デフォルトで HTTPS ポート番号は 443 です。ポート番号に関する詳細については、[HTTP ポート](#) を参照ください。

非動作プロセスのタイムアウト

設定できる場所	名称	コメント
webServer オブジェクト	inactiveProcessTimeout	
WEB SET OPTION	Web inactive process timeout	
設定ダイアログボックス	オプション (I) ページ / 非動作プロセスのタイムアウト	スライダー

セッションと紐づいた非アクティブ Web プロセスのタイムアウト時間 (分単位) を設定します。すると、On Web Close Process データベースメソッドが呼び出され、セッションのコンテキストは削除されます。

デフォルト値: 480 分 (デフォルト値に戻すには 0 を指定します)

非アクティブセッションタイムアウト

設定できる場所	名称	コメント
webServer オブジェクト	inactiveSessionTimeout	
WEB SET OPTION	Web inactive session timeout	

非アクティブセッションのタイムアウト時間 (分単位) を cookie に設定します。タイムアウト時間が経過するとセッション cookie が無効になり、HTTP クラリアントによって送信されなくなります。

デフォルト値: 480 分 (デフォルト値に戻すには 0 を指定します)

リクエストを受け付ける IP アドレス

設定できる場所	名称	コメント
webServer オブジェクト	IPAddressToListen	
WEB SET OPTION	Web IP address to listen	
設定ダイアログボックス	設定ページ / IPアドレス	ポップアップメニュー

4D Webサーバーが HTTPリクエストを受け付ける IPアドレスを指定できます (4Dローカルおよび 4D Server)。

デフォルトでは、特定のアドレスが定義されていないため (設定ダイアログボックスでは 任意 の値)、サーバーはすべての IPアドレスに応答します。特定のアドレスを指定すると、サーバーはこの IPアドレスへのリクエストにのみ応答します。この機能は複数の TCP/IPアドレスが設定されたマシン上で動作する 4D Webサーバーのためのものです。これはしばしば、インターネットホストプロバイダーで使用されます。

とりうる値: IPアドレス文字列。IPv6 文字列フォーマット (例: "2001:0db8:0000:0000:0000:ff00:0042:8329") と IPv4 文字列フォーマット (例: "123.45.67.89") の両方がサポートされます。

IPv6 のサポートについて

- TCPポートが使用済みでも警告は出ません
サーバーが応答する IPアドレスが "任意" に設定されていた場合、TCPポートが他のアプリケーションで使用されていても、それはサーバー起動時に指摘されません。IPv6 アドレスのポートが空いているため、この場合 4D Server はどのようなエラーも検知しません。しかしながら、マシンの IPv4 アドレスを使用、またはローカルアドレス 127.0.0.1 を使用してアクセスすることは不可能です。
定義されたポートで 4D Server が反応していないようであれば、サーバーマシンで [::1] のアドレスを試してみてください (IPv6 における 127.0.0.1 と同義です。他のポート番号をテストするには [:portNum] を追加してください)。4D が応答するようであれば、IPv4 のポートを他のアプリケーションが使用している可能性が高いです。
- IPv4-マップされた IPv6アドレス
プロセスを標準化するために、4D では IPv4アドレスの標準ハイブリッド表示を IPv6 で提供しています。これらのアドレスは IPv6フォーマットにおいて 96ビットの接頭辞付きで書かれ、その後に IPv4ドット区切り表記で書かれた 32ビットが続きます。たとえば、::ffff:192.168.2.34 は、192.168.2.34 という IPv4アドレスを表します。
- ポート番号の表記
IPv6 記法はコロン (:) を使用するので、ポート番号を追加するときには混乱を招く恐れがあることに注意が必要です。たとえば:

```
2001:0DB8::85a3:0:ac1f:8001 // IPv6アドレス
2001:0DB8::85a3:0:ac1f:8001:8081 // ポート 8081 指定の IPv6アドレス
```

混乱を避けるため、IPv6アドレスをポート番号と併記する際には、以下の様に [] でアドレスを囲う記法が推奨されます:

```
[2001:0DB8::85a3:0:ac1f:8001]:8081 // ポート 8081 指定の IPv6アドレス
```

旧式セッション (自動セッション管理)

設定できる場所	名称	コメント
webServer オブジェクト	keepSession	
WEB SET OPTION	Web keep session	
設定ダイアログボックス	オプション (I) ページ / 旧式セッション (シングルプロセスセッション)	変換されたプロジェクトのみ

4D Webサーバーによる旧式セッション管理を有効/無効にします (廃止予定)。

このオプションが選択されていると、"一時的なコンテキストを再利用する" オプションも自動で選択され、ロックされます。

ログの記録

設定できる場所	名称	コメント
webServer オブジェクト	logRecording	
WEB SET OPTION	<code>Web log recording</code>	
設定ダイアログボックス	ログ (タイプ) ページ	popupアップメニュー

4D Web サーバーが受け取るリクエストのログを開始/停止します。ログは、*logweb.txt* ファイルに記録され、そのフォーマットを指定することができます。デフォルトでは、リクエストは規則されません (0 / ログファイルなし)。有効化されると、*logweb.txt* ファイルが Logs フォルダー内に自動で保存されます。

このファイルのフォーマットを指定することができます。使用可能な値:

値	フォーマット	説明
0	ログファイルなし	デフォルト
1	CLF形式で記録する	Common Log Format - それぞれのリクエストが行単位でファイル内に表示されます: <code>host rfc931 user [DD/MMM/YYYY:HH:MM:SS] "request" state length</code> - 各フィールドはスペースによって区切られ、各行は CR/LF シーケンスで終ります。
2	DLF形式で記録する	Combined Log Format - CLFフォーマットを使いながら、各リクエストの最後に 2つのHTTPフィールド、RefererとUser-agent を追加します。
3	ELF形式で記録する	Extended Log Format - 設定ダイアログボックスにてカスタマイズします。
4	WLF形式で記録する	WebStar Log Format - 設定ダイアログボックスにてカスタマイズします。

フォーマット3、4はカスタムフォーマットで、あらかじめ [設定ダイアログボックス](#) にて内容を指定しておく必要があります。このページでフィールドを選択せずにこれらのフォーマットを使用した場合、ログファイルは生成されません。

最大同時Webプロセス

設定できる場所	名称	コメント
webServer オブジェクト	maxConcurrentProcesses	
WEB SET OPTION	<code>Web max concurrent processes</code>	
設定ダイアログボックス	オプション (I) ページ / 最大同時Webプロセス	

セッションなし または 旧式セッション が使用されているときの、サーバー上で同時に開くことのできるすべての Webプロセスの最大同時接続数の厳格な上限を設定します (スケーラブルセッション では、[数の制限なく](#) プリエンブティップロセスがサポートされています)。このパラメーターは、異常な数のリクエストによる 4D Webサーバーの飽和状態を避けるために使用します。最大Web同時接続数 (マイナス1) に達すると、4D は新しいプロセスを作成せず、HTTPステータス 503 – Service Unavailable を新規リクエストに対して返信します。

デフォルト値は 100 です。10から32000までの値を設定できます。

最大リクエストサイズ

設定できる場所	名称	コメント
webServer オブジェクト	<code>maxRequestSize</code>	
WEB SET OPTION	<code>Web maximum requests size</code>	

Webサーバーに処理を許可する HTTPリクエスト (POST) の最大サイズ (バイト単位)。デフォルト値は 2,000,000 (2MBより、すこし少ない値) です。最大値 (2,147,483,648) に設定した場合、実際には制限無しということになります。

制限を設けることで、サイズが非常に大きいリクエストによって Webサーバーが過負荷状態に陥ることを防ぎます。制限を設けることで、サイズが非常に大きいリクエストによって Webサーバーが過負荷状態に陥ることを防ぎます。

とりうる値: 500,000 - 2,147,483,648。

最大同時セッション数

設定できる場所	名称	コメント
webServer オブジェクト	<code>maxSessions</code>	
WEB SET OPTION	<code>Web max sessions</code>	

同時セッション上限数。上限に達すると、Webサーバーが新規セッションを作成するときに、一番古いセッションが閉じられます (On Web Close Process データベースメソッドが呼び出されます)。同時セッション数は、[Webプロセスの最大値](#)を超えることはできません (デフォルトは 100)。

デフォルト値: 100 (デフォルト値に戻すには 0 を指定します)。

最低TLSバージョン

設定できる場所	名称	コメント
webServer オブジェクト	<code>minTLSVersion</code>	number

接続に必要な最低TLSバージョン。これよりも低いバージョンのみをサポートするクライアントからの接続は拒否されます。

とりうる値:

- 1 = TLSv1_0
- 2 = TLSv1_1
- 3 = TLSv1_2 (デフォルト)
- 4 = TLSv1_3

変更した場合、設定を反映するには Webサーバーを再起動する必要があります。

4D が使用する最低TLSバージョンは、`SET DATABASE PARAMETER` コマンドを使用してセッションごとに変更することができます。この場合、変更は 4Dアプリケーション全体に適用されます (Webサーバー・SQLサーバー・クライアント/サーバー接続を含む)。

名称

設定できる場所	名称	コメント
webServer オブジェクト	<code>name</code>	

Webサーバーアプリケーションの名称。コンポーネントの Webサーバーが起動されているときに便利です。

OpenSSL バージョン

設定できる場所	名称	コメント
webServer オブジェクト	openSSLVersion	読み取り専用

使用されている OpenSSLライブラリのバージョン。

Perfect Forward Secrecy (PFS)

設定できる場所	名称	コメント
webServer オブジェクト	perfectForwardSecrecy	ブール; 読み取り専用

Webサーバーの PFS利用可否状況 ([TLS 参照](#))。

一時的なコンテキストを再利用する (リモートモード)

設定できる場所	名称	コメント
設定ダイアログボックス	オプション (I) ページ / 最大同時Webプロセス	

このオプションは、セッションなし オプションがチェックされている場合にのみ利用できます。

前の Webリクエストを処理するために作成された Webプロセスを再利用することによって、4Dリモートモードで実行されている 4D Webサーバーの動作を最適化できます。実際、4D Webサーバーはそれぞれの Webリクエストを処理するために専用の Webプロセスを必要とします。リモートモードでは、このプロセスは必要に応じて、データやデータベースエンジンにアクセスするために 4D Server に接続します。そしてプロセス独自の変数やセレクションを使用して、一時的なコンテキストを作成します。リクエストの処理が終了すると、このプロセスは廃棄されます。

一時的なコンテキストを再利用するオプションがチェックされていると、リモートモードの 4D は作成された固有の Webプロセスを保守し、その後のリクエストで再利用します。プロセスの作成処理が省略されるため、Webサーバーのパフォーマンスが向上します。

他方このオプションを使用する場合、不正な結果が返されることを避けるために、4Dメソッド内で使用される変数をシステムチックに初期化する必要があります。同様に、以前のリクエストで使用されたカレントセレクションやカレントレコードをアンロードする必要があります。

このオプションはリモートモードの 4D Webサーバーでのみ効果があります。ローカルモードの 4D では (セッション管理をおこなうプロセスを除く) すべてのWebプロセスが使用後に終了されます。

Robots.txt

特定のクローラー (クエリエンジン、スパイダー...) は Webサーバーやスタティックページをクロールします。クローラーにサイトへアクセスさせたくない場合、アクセスを禁止する URL を指定できます。

これには、ROBOTS.TXT ファイルをサーバーのルートに置きます。このファイルの内容は以下の構造になっていなければなりません:

```
User-Agent: <name>
Disallow: <URL> または <beginning of the URL>
```

たとえば:

```
User-Agent: *
Disallow: /4D
Disallow: /%23%23
Disallow: /GIFS/
```

- "User-Agent: *" は、すべてのクローラーが対象であることを示します。
- "Disallow: /4D" は、/4D から始まる URL へのアクセスを許可しないことをクローラーに通知します。

- "Disallow: /%23%23" は、/%23%23 から始まる URL へのアクセスを許可しないことをクローラーに通知します。
- "Disallow: /GIFS/" は、/GIFS/ フォルダーおよびそのサブフォルダーへのアクセスを許可しないことをクローラーに通知します。

他の例題:

```
User-Agent: *
Disallow: /
```

この場合、クローラーにサイト全体へのアクセスを許可しないことを通知します。

ルートフォルダー

設定できる場所	名称	コメント
webServer オブジェクト	<code>rootFolder</code>	テキストプロパティ (<code>start()</code> 関数の <code>settings</code> パラメーターと使用する場合 は、 <code>4D.Folder</code> オブジェクトも使用可能)
WEB SET ROOT FOLDER		
設定ダイアログボックス	設定ページ / デフォルト HTMLルート	

4D がブラウザーに送信するスタティック/セミダイナミックな HTMLページ、ピクチャーなどを検索するフォルダーを指定します。これが、Webサーバーのルートフォルダーです。パスは、POSIXフルパスの形式です。ルートフォルダーの変更を反映するには、Webサーバーを再起動する必要があります。

さらに HTMLルートフォルダーは、Webサーバーのディスク上で、ファイルに対するアクセスができない階層を定義することになります。ブラウザーから送られた URL や 4Dコマンドが、HTMLルートフォルダーよりも上の階層にアクセスしようとすると、ファイルが存在しないことを示すエラーが返されます。

デフォルトで、4D は WebFolder という名前のデフォルトHTMLルートフォルダーを定義します。Webサーバーの初回起動時にこのフォルダーが存在しなければ、HTMLルートフォルダーは物理的にディスク上に作成されます。ルートフォルダーは以下の場所に作成されます:

- 4D (ローカル) および 4D Server では、[Project フォルダー](#) と同階層。
- 4Dリモートモードでは、ローカルのリソースフォルダー内

デフォルトHTMLルートフォルダーを変更するには、パスを "デフォルトHTMLルート" に入力します。

- このとき、相対パスの起点は [Project フォルダー](#) (4Dローカルおよび 4D Server)、または、4Dアプリケーションやソフトウェアーパッケージを含むフォルダーです (4Dリモートモード)。
- パスは POSIX シンタックスで表します (フォルダーはスラッシュ ("/") で区切れます)。
- フォルダー階層で 1つ上にあがるには、フォルダー名の前にピリオドを2つ ".." 置きます。
- パスはスラッシュ ("/") で始まつてはいけません (HTMLルートフォルダーを Projectフォルダーや 4Dリモートフォルダーにしながら、それより上階層へのアクセスを禁止したい場合には、"/" を入力します)。

たとえば、HTMLルートフォルダーを "MyWebApp" フォルダーの "Web" サブフォルダーにしたい場合、"MyWebApp/Web" と入力します。

HTMLルートフォルダーを変更すると、アクセスが制限されているファイルを格納しないようにするために、キャッシュがクリアされます。

スケーラブルセッション

設定できる場所	名称	コメント
webServer オブジェクト	<code>scalableSession</code>	
WEB SET OPTION	Web スケーラブルセッション	
設定ダイアログボックス	オプション (I) ページ / スケーラブルセッション (マルチプロセスセッション)	

4D Webサーバーでのスケーラブルセッション管理を有効/無効にします。Webサーバーセッションの詳細については、[ユーザーセッション](#) のページを参照ください。

セッションcookieドメイン

設定できる場所	名称	コメント
webServer オブジェクト	<code>sessionCookieDomain</code>	
WEB SET OPTION	<code>Web session cookie domain</code>	

セッションcookie の "path" フィールド。セッションcookie のスコープを制御するのに使用されます。たとえば、このセレクターに "/4DACTION" という値を設定した場合、4DACTION で始まる動的リクエストの場合にのみクライアントは cookie を送信し、ピクチャーや静的ページへのリクエストは除外されます。

セッションcookie名

設定できる場所	名称	コメント
webServer オブジェクト	<code>sessionCookieName</code>	
WEB SET OPTION	<code>Web session cookie name</code>	

セッションID の保存に使用されるセッションcookie の名称。デフォルト = "4DSID"。

セッションcookieパス

設定できる場所	名称	コメント
webServer オブジェクト	<code>sessionCookiePath</code>	
WEB SET OPTION	<code>Web session cookie path</code>	

セッションcookie の "domain" フィールドの値。セッションcookie のスコープを制御するのに使用されます。たとえば、このセレクターに "/*.4d.fr" の値を設定した場合、リクエストの宛先が ".4d.fr" のドメインに限り、クライアントは cookie を送信します。

セッションcookie SameSite

設定できる場所	名称	コメント
webServer オブジェクト	<code>sessionCookieSameSite</code>	

セッションcookie の `SameSite` 属性の値。この属性は、一部のクロスサイトリクエストフォージェリ (CSRF) 攻撃からの保護として、ファーストパーティーコンテキストまたは同一サイトコンテキストのどちらかに cookie を限定するかを宣言することができます。

SameSite 属性に関する詳細な説明は [Mozilla のドキュメンテーション](#) または [こちらの Web開発ページ \(英語\)](#) を参照ください。

次の値が提供されています:

- "Strict" (4Dセッションcookie の `SameSite` 属性のデフォルト値): ファーストパーティーコンテキスト、すなわち現在のサイトのドメインに一致するコンテキストでのみ cookie は送信され、サードパーティの Webサイトには決して送信されません。
- "Lax": クロスサイトのサブリクエストでは cookie は送信されませんが (たとえば、画像やフレームをサードパーティのサイトにロードする場合など)、ユーザーがオリジンのサイトに移動するとき (つまり、リンクを辿っているとき) には送信されます。
- "None": ファーストパーティーやオリジン間リクエストにかかわらず、すべてのコンテキストにおいて cookie が送信されます。"None" を使用する場合は、cookie の `Secure` 属性も設定する必要があります (設定しないと、cookie がブロックされます)。

セッションcookie の `Secure` 属性値は、HTTPS接続の場合には (`SameSite` 属性値が何であれ)、自動的に "True" に設定されます。

HTTPサーバーで `SameSite=None` を設定することは、(HTTPS でのみ使用される) `Secure` 属性が欠落し、cookie がブロックされるため、推奨されません。

プリエンプティブプロセスを使用

設定できる場所	名称	コメント
設定ダイアログボックス	オプション (I) ページ / 最大同時Webプロセス	

このオプションは、セッションなし オプションが選択されている場合に、アプリケーションの Webサーバーコードのプリエンプティブモードを有効にします（スケーラブルセッション では、プリエンプティブモードは常に有効です）。このコンテキストにおいて当該オプションがチェックされているとき、4Dコンパイラは [Web関連のコード](#) それぞれのスレッドセーフプロパティを自動的に評価し、違反があった場合にはエラーを返します。

廃止予定の設定

以下の設定は現在もサポートされていますが、廃止予定の機能や技術に依存しています。通常はデフォルト値のままにしておくことが推奨されます。

"4DSYNC" URLを使用したデータベースアクセスを許可

このオプションを使用して、廃止予定の `/4DSYNC URL` による HTTP同期サポートを制御します。

セッション IPアドレス検証

スケーラブルセッションモード の場合には、このオプションは利用できません（検証はおこなわれません）。

セッションcookie の IP アドレス検証のステータス。セキュリティ上の理由により、セッションcookie を持つ各リクエストに対して 4D Webサーバーはデフォルトで IPアドレスを検証します。アプリケーションによっては、この検証機能を無効化し、IPアドレスが合致しなくてもセッションcookie を受け入れるようにしたいかもしれません。たとえば、モバイルデバイスが WiFi と 4G/5G ネットワークを切り替えた場合、IPアドレスが変更されます。このように IPアドレスが変更しても、クライアントによる Webセッションの継続を許可するには、このオプションに 0 を渡します。この設定はアプリケーションのセキュリティレベルを下げることに留意が必要です。この設定が変更された際には、その設定は直ちに反映されます（HTTPサーバーを再起動する必要はありません）。

拡張文字をそのまま送信

このオプションを有効にすると、Webサーバーはセミダイナミックページの拡張文字を、HTML標準に基づいた変換をおこなわずに「そのまま」送信します。このオプションにより、とくに Shift_JIS文字コード利用時の日本語のシステムで速度が向上します。

Keep-Alive接続を使用する

4D Webサーバーは Keep-Alive接続を使用できます。keep-alive接続を使用すると、Webブラウザーとサーバー間の一連のやり取りについて単一のTCP接続を維持し、システムリソースの節約と通信の最適化を図ることができます。

Keep-Alive接続を使用する オプションは、Webサーバーの Keep-Alive接続を有効および無効にします。このオプションはデフォルトで有効になっています。ほとんどの場合、通信が高速化されるため、この状態をお勧めします。Webブラウザーが Keep-Alive接続をサポートしない場合、4D Webサーバーは自動で HTTP/1.0 に切り替えます。

4D Webサーバーの Keep-Alive機能はすべての TCP/IP接続 (HTTP, HTTPS) に関連します。しかしながら、すべての 4D Webプロセスで常に Keep-Alive接続が使用されるわけではないことに留意してください。

あるケースでは、内部的な他の最適化機能が呼び出されることがあります。Keep-Alive接続は、おもにスタティックページで有効です。

Keep-Alive接続を設定する 2つのオプションがあります：

- 接続毎のリクエスト数：ひとつの Keep-Alive接続におけるリクエストとレスポンスの最大数を設定します。接続あたりのリクエスト数を制限することで、サーバーのリクエスト過多を避けることができます（攻撃者が使用するテクニック）。

4D Webサーバーをホストするマシンのリソースに応じて、デフォルト値 (100) を増減できます。

- タイムアウト：この値を使用して、Webブラウザーからリクエストがおこなわれない状態で、Webサーバーが開かれた接続を保守する最大の待ち秒数を設定します。この秒数が経過すると、サーバーは接続を閉じます。

接続が閉じられた後に Webブラウザーがリクエストを送信すると、新しい TCP接続が作成されます。この動作はユーザーからは見えません。

管理

4Dには、統合された Webサーバーを起動・停止・監視するためのツールがいくつか用意されています。

4D Webサーバーの開始

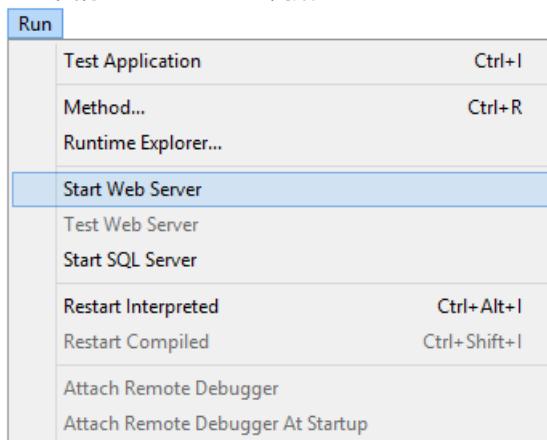
4D や 4D Server の Webサーバーを起動するには、"4D Web Application" ライセンスが必要です。詳細については [4D Webサイト](#) を参照ください。

4Dプロジェクトは、メイン（ホスト）アプリケーションおよび、ホストされた各コンポーネントの Webサーバーを起動して監視することができます。

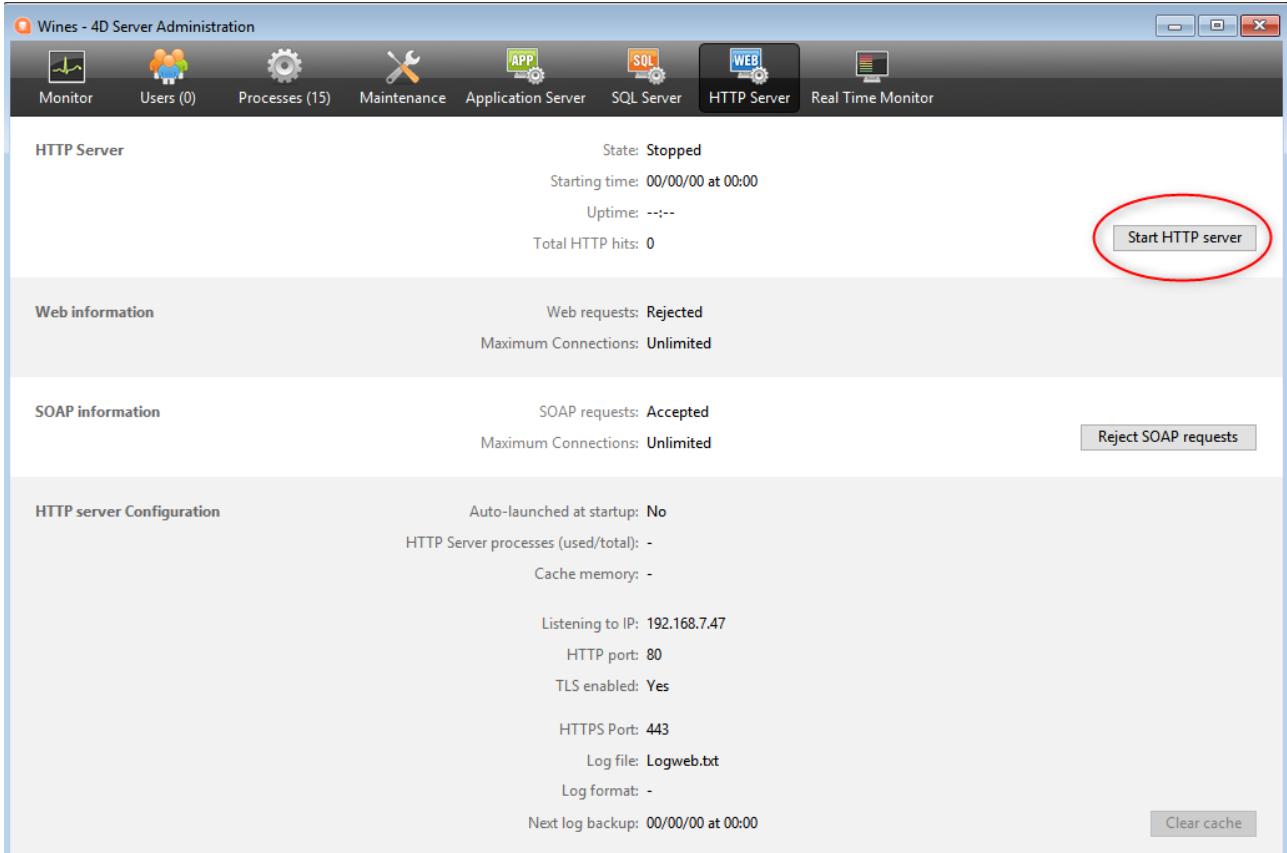
4D Webサーバーは複数の方法で起動できます：

- ボタン/メニュー命令の使用。

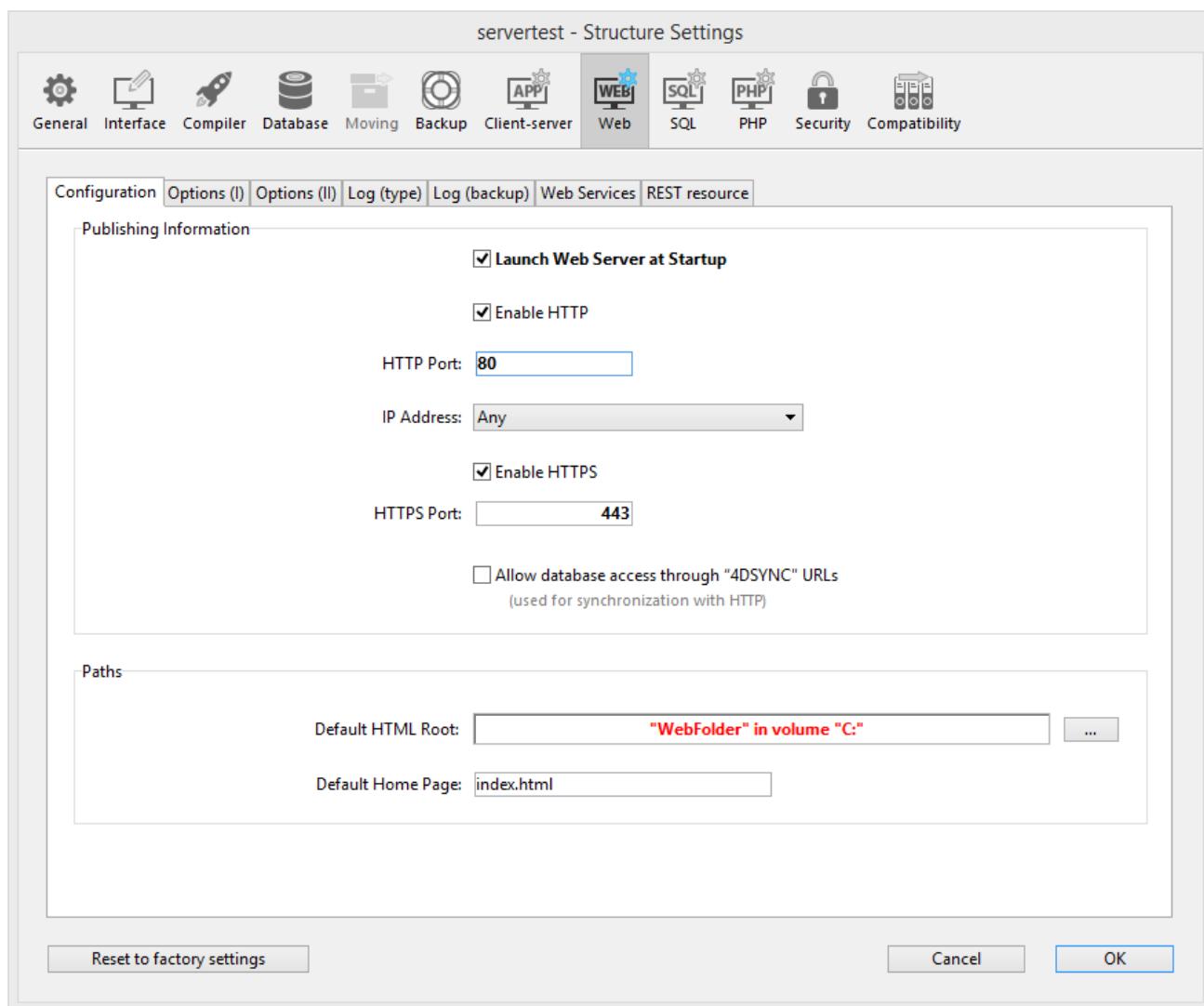
- 4D: 実行 > Webサーバー開始 メニュー



- 4D Server: HTTPサーバーページの HTTPサーバー開始 ボタン



- 4Dアプリケーション開始時に Webサーバーを自動起動。これには、ストラクチャー設定の Web/設定ページを表示し、開始時にWebサーバーを起動 オプションを有効にします:



- `webServer.start()` 関数または `WEB START SERVER` コマンドを呼び出してプログラムで開始。

コンポーネントの Webサーバーは、コンポーネントの WebServer オブジェクトに対して `webServer.start()` 関数を呼び出すことで開始できます。

Webサーバーを開始したり停止したりするために、4Dアプリケーションを再起動する必要はありません。

4D Webサーバーの停止

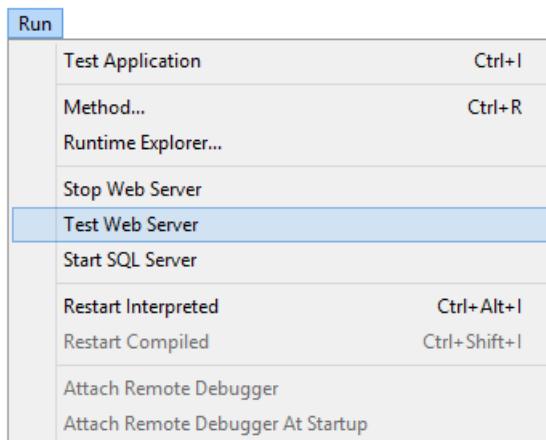
4D Webサーバーは複数の方法で停止できます:

- 4D の 実行 > Webサーバー停止 メニューを使用するか、4D Server にて HTTPサーバーページの HTTPサーバー停止 ボタンを使用する (いずれも、サーバー開始前は …開始 と表示されています)。
- `webServer.stop()` 関数または `WEB STOP SERVER` コマンドを呼び出してプログラムで停止。

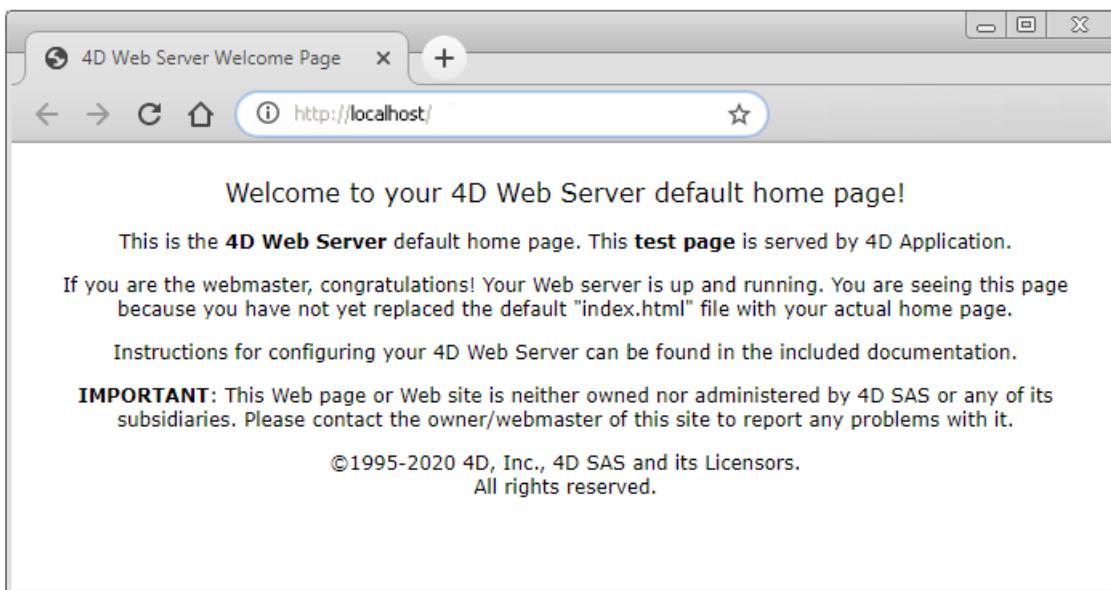
コンポーネントの Webサーバーは、コンポーネントの WebServer オブジェクトに対して `webServer.stop()` 関数を呼び出すことで停止できます。

4D Webサーバーのテスト

Webサーバーテスト メニューコマンドを使用してビルトインの Webサーバーが正しく実行されているか確認できます (4Dのみ)。このメニューは Webサーバーが実行されているときに 実行 メニューからアクセスできます:



このコマンドを選択すると、4Dアプリケーションが公開しているWebサイトのホームページが、デフォルトWebブラウザに表示されます：



このコマンドでWebサーバーの動作や、ホームページの表示などを検証できます。ページは、Webブラウザが実行されているマシンのIPアドレスを指定する標準のショートカットである、ローカルホストのURLを使用して呼び出されます。コマンドはストラクチャー設定で指定された [TCP公開ポート](#) 番号を考慮に入れます。

キャッシュクリア

いつでもページやイメージをキャッシュからクリアできます（たとえば、スタティックページを更新し、キャッシュにそれをリロードさせたい場合）。

これをおこなうには：

- 4D: ストラクチャー設定の Web / オプション (I) ページの キャッシュクリア ボタンをクリックします。
- 4D Server: [4D Server 管理ウィンドウ](#) の HTTPサーバーページにて、キャッシュクリア ボタンをクリックします。

キャッシュは即座にクリアされます。

特殊なURL </4DCACHECLEAR> を使用することもできます。

ランタイムエクスプローラー

Webサーバーに関連する情報は、ランタイムエクスプローラーにある ウオッч ページ (Web 項目内) に表示されます。

- Webキャッシュ使用：Webキャッシュに存在するページ数とその使用率を示します。Webサーバーがアクティブでキャッシュサイズが 0 より大きい場合のみ、この情報が利用できます。
- Webサーバー経過時間：Webサーバーの使用時間を（時間：分：秒 フォーマットで）示します。Webサーバーがアクティブである場合のみ、この情報が利用できます。

- Webヒット数：Webサーバーが起動してから受け取った HTTPリクエストの総数と、毎秒のリクエスト数を示します（ランタイムエクスプローラーの更新の間で測定）。Webサーバーがアクティブである場合のみ、この情報が利用できます。

管理用 URL

Webサイト管理用の URL を使用して、サーバー上に公開している Webサイトをコントロールできます。4D Webサーバーは、/4DSTATS、/4DHTMLSTATUS、/4DCACHECLEAR と /4DWEBTEST の 4つの URL を受け入れます。

/4DSTATS、/4DHTMLSTATUS と /4DCACHECLEAR はデータベースの設計者と管理者のみが利用可能です。4D のパスワードシステムが起動されていないと、これらの URL はすべてのユーザーに対して利用可能となります。/4DWEBTEST は、常に利用可能です。

/4DSTATS

/4DSTATS URL は以下の情報を（ブラウザーで表示可能な）HTML の表形式で返します：

項目	説明
現在のキャッシュサイズ	Webサーバーの現在のキャッシュサイズ（バイト単位）
最大キャッシュサイズ	キャッシュの最大サイズ（バイト単位）
キャッシュされたオブジェクトの最大サイズ	キャッシュされたオブジェクト中で最も大きなもの（バイト単位）
使用キャッシュ	キャッシュ使用率
キャッシュされているオブジェクト	キャッシュされているオブジェクトの数（ピクチャー含む）。

この情報を用いて、サーバーの機能を確認することができ、最終的には対応するパラメーターを適合させます。

WEB GET STATISTICS コマンドを使用して、スタティックページに対してキャッシュがどのように使用されているかに関する情報を入手することができます。

/4DHTMLSTATUS

/4DHTMLSTATUS URL は、/4DSTATS URLと同じ情報を HTML表形式で返します。その違いは キャッシュされているオブジェクト に HTMLページの情報のみが返され、ピクチャーファイルをカウントしないことです。さらにこの URL は フィルターされたオブジェクト の情報を返します。

項目	説明
現在のキャッシュサイズ	Webサーバーの現在のキャッシュサイズ（バイト単位）
最大キャッシュサイズ	キャッシュの最大サイズ（バイト単位）
キャッシュされたオブジェクトの最大サイズ	キャッシュされたオブジェクト中で最も大きなもの（バイト単位）
使用キャッシュ	キャッシュ使用率
キャッシュされているオブジェクト	キャッシュされているオブジェクトの数（ピクチャーを除く）。
フィルターされたオブジェクト	URL でカウントされないキャッシュ中のオブジェクトの数（特にピクチャー）。

/4DCACHECLEAR

/4DCACHECLEAR URLは、スタティックページとイメージのキャッシュを即座に消去します。そのため、修正されたページを "強制的に" 更新することができます。

/4DWEBTEST

/4DWEBTEST URLは、Webサーバーの状態を確認するために設計されています。このURLが呼び出されると、4D は以下の HTTPフィールドを記したテキストファイルを返します。

HTTP フィールド	説明	例題
日付	RFC 822 フォーマットでの現在の日付	Mon, 7 Dec 2020 13:12:50 GMT
Server	4D/バージョン番号	4D/18.5.0 (Build 18R5.257368)
User-Agent	名前とバージョン @ IP クライアントアドレス	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.183 Safari/537.36 @ 127.0.0.1

Logs

4Dでは、Webリクエストのログを2種類作成することができます:

- Webサーバーの開発段階で有用なデバッグルог (HTTPDebugLog.txt)。
- おもに統計目的で使用される、標準化された Webリクエストログ (logweb.txt)。

両方のログファイルは、アプリケーションプロジェクトの Logs フォルダーに自動的に作成されます。

HTTPDebugLog.txt

[WebServer オブジェクト](#) または `WEB SET OPTION` コマンドを使って、[http デバッグファイル](#) を有効化することができます。

このログファイルは、各 HTTPリクエストとそれぞれのレスポンスを rawモードで記録します。ヘッダーを含むリクエスト全体が記録され、オプションでボディ部分も記録することができます。

リクエストとレスポンスの両方に対して以下のフィールドが記録されます:

フィールド名	説明
SocketID	通信に使用されたソケットの ID
PeerIP	ホスト (あるいはクライアント) の IPv4アドレス
PeerPort	ホスト (あるいはクライアント) が使用したポート番号
TimeStamp	(システムが開始されてからの) ミリ秒単位でのタイムスタンプ
ConnectionID	接続UUID (通信に使用された VTCPSocket の UUID)
SequenceNumber	ログセッション内で固有かつシーケンシャルなオペレーション番号

logweb.txt

[WebServer オブジェクト](#)、`WEB SET OPTION` コマンド、またはストラクチャー設定の Web/ログ (タイプ) ページを使って、[Webログファイル](#) を有効化することができます。ログのフォーマットを選択する必要があります。

CLF/DLF

それぞれのリクエストが行単位でファイル内に表示されます: `host rfc931 user [DD/MMM/YYYY:HH:MM:SS] "request" state length`。各フィールドはスペースによって区切られ、各行は CR/LF シーケンス (character 13, character 10) で終ります。

DLF (Combined Log Format) フォーマットは CLF (Common Log Format) フォーマットと類似していて、まったく同じ構造を使用します。さらに、各リクエストの最後に2つのHTTPフィールド、Referer と User-agent を追加します。CLF/DLF フォーマット (カスタマイズ不可) についての説明です:

フィールド名	説明
host	クライアントの IPアドレス (例: 192.100.100.10)
rfc931	4Dによって作成されない情報。常に - (マイナス記号) です。
user	認証されているユーザー名、あるいは、- (マイナス記号) 。ユーザー名にスペースが含まれると、_ (下線) に置き換わります。
DD/MMM/YYYY:HH:MM:SS	DD: 日、MMM: 月を表す3文字の略号 (Jan, Febなど) 、YYYY: 年、HH: 時間、MM: 分、SS: 秒。日付と時間はサーバーのローカルタイム。
request	クライアントによって送られたリクエスト (例: GET /index.htm HTTP/1.0) 。
state	サーバーの返答。
length	返されたデータ (HTTPヘッダー以外) のサイズまたは 0
Referer	DLF のみ - リクエストされたドキュメントを指しているページの URL を含みます。
User-agent	DLF のみ - リクエストのオリジンにおけるクライアントのブラウザーまたはソフトウェアの名前とバージョンを含みます。

ELF/WLF

ELF (Extended Log Format) フォーマットは HTTPブラウザー界で広く普及しています。そして、特別なニーズに応える洗練されたログを構築します。この理由により、ELFフォーマットはカスタマイズされます。記録するフィールドやそのフィールドをファイルへ挿入する順番を選択することができます。

WLF (WebStar Log Format) フォーマットは 4D WebSTAR サーバー用として特別に開発されました。

フィールドの設定

ELF または WLF を選択すると、選択されたフォーマットに対して利用可能なフィールドが "Weg Log Token Selection" エリアに表示されます。ログに含む各フィールドを選択する必要があります。これには、選択するフィールドにチェックを入れます。

同じフィールドを 2度選択することはできません。

各フォーマットで利用可能なフィールド (アルファベット順) とその内容を以下のテーブルに示します:

フィールド	ELF	WLF	値
BYTES_RECEIVED		○	サーバーが受け取ったバイト数
BYTES_SENT	○	○	サーバーがクライアントに送ったバイト数
C_DNS	○	○	DNSのIPアドレス (ELF: C_IP フィールドと同一のフィールド)
C_IP	○	○	クライアントの IPアドレス (例: 192.100.100.10)
CONNECTION_ID		○	接続ID番号
CS(COOKIE)	○	○	HTTPリクエストに格納されている cookies に関する情報
CS(HOST)	○	○	HTTPリクエストの Hostフィールド
CS(REFERER)	○	○	リクエストされたドキュメントを指すページの URL
CS(USER_AGENT)	○	○	ソフトウェアとクライアントのオペレーティングシステムに関する情報
CS_SIP	○	○	サーバーの IPアドレス
CS_URI	○	○	リクエストが作成された URI
CS_URI_QUERY	○	○	リクエストのクエリ引数
CS_URI_STEM	○	○	クエリ引数のないリクエストのパート
DATE	○	○	DD: 日、MMM: 月を表す3文字の略号 (Jan, Febなど)、YYYY: 年
METHOD	○	○	サーバーへ送られたリクエスト用の HTTPメソッド
PATH_ARGS		○	CGI引数: "\$" の後に続く文字列
STATUS	○	○	サーバーの返答
TIME	○	○	HH: 時間、MM: 分、SS: 秒
TRANSFER_TIME	○	○	返答を作成するためにサーバーが要求した時間
USER	○	○	認証されているユーザー名、あるいは、- (マイナス記号)。ユーザー名にスペースが含まれると、_ (下線) に置き換わります。
URL		○	クライアントがリクエストした URL

日付と時間は GMTで表されます。

周期的なバックアップ

logweb.txt ファイルはかなり膨大になることがあるため、自動のアーカイブメカニズムを構築することができます。バックアップはある周期 (時間、日、週、月単位) または、ファイルのサイズに基づいて起動します。設定の期限 (またはファイルサイズ) に近づくと、4D は自動的にカレントのログファイルを閉じてアーカイブします。そして新たにファイルを作成します。

Web のログファイル用のバックアップが起動すると、ログファイルは "Logweb Archives" という名前のフォルダーにアーカイブされます。このフォルダーは、*logweb.txt* ファイルと同じ階層に作成されます。

アーカイブされたファイルは、以下の例に基づいて名称変更されます: "DYYYY_MM_DD_Thh_mm_ss.txt"。たとえば、ファイルがアーカイブされた時間が September 4, 2020 at 3:50 p.m. and 7 seconds である場合、"D2020_09_04_T15_50_07.txt" になります。

バックアップパラメーター

logweb.txt の自動バックアップパラメーターは、ストラクチャー設定の Web/ログ (バックアップ) ページで設定します:

webServer - Structure Settings

Backup Frequency for Web Log File

- No Backup
- Every hour(s) starting at
- Every day(s) at
- Every week(s)
 - Monday at
 - Tuesday at
 - Wednesday at
 - Thursday at
 - Friday at
 - Saturday at
 - Sunday at
- Every month(s) Day at
- Every

[Reset to factory settings](#) [Cancel](#) [OK](#)

最初に、頻度（日、週などの単位）またはファイルサイズの上限に対応するラジオボタンをクリックして選択します。必要に応じて、バックアップする正確な時間を指定します。

- バックアップしない：周期的なバックアップ機能が無効になっています。
- X 時間ごと：1時間単位でバックアップをプログラムする際、このオプションを使用します。1から24の値を入力します。
 - 開始時刻：最初のバックアップ開始時間の設定に使用します。
- X 日ごと：1日単位でバックアップをプログラムする際、このオプションを使用します。バックアップを毎日実行するには、1を入力します。このオプションをチェックすると、バックアップの開始時間を指定しなければなりません。
- X 週ごと：1週間単位でバックアップをプログラムする際、このオプションを使用します。たとえば、毎週バックアップをおこなうには1と設定します。たとえば、毎週バックアップをおこなうには1と設定します。このオプションをチェックすると、バックアップを開始する曜日と時間を指定しなければなりません。複数の曜日を選択することもできます。
- X 月ごと：1ヶ月単位でバックアップをプログラムする際、このオプションを使用します。たとえば、毎月バックアップをおこなうには1と設定します。たとえば、毎月バックアップをおこなうには1と設定します。
- X MB (サイズ指定)：カレントのリクエストログのファイルサイズに基づいてバックアップをプログラムする際、このオプションを使用します。ファイルが指定サイズに達すると、バックアップが自動的に起動します。サイズ制限は1、10、100または1000MBごとに設定可能です。

Webサーバーオブジェクト

4Dプロジェクトは、メイン（ホスト）アプリケーションおよび、ホストされた各コンポーネントの Webサーバーを起動して監視することができます。

たとえば、メインアプリケーションに 2つのコンポーネントをインストールしている場合、アプリケーションから最大 3つの独立した Webサーバーを起動して監視することができます：

- ホストアプリケーションの Webサーバーを1つ
- コンポーネント#1 の Webサーバーを1つ
- コンポーネント#2 の Webサーバーを1つ

1つの 4Dアプリケーションプロジェクトに接続できるコンポーネントの数、つまり Webサーバーの数には、メモリ以外の制限はありません。

メインアプリケーションの Webサーバーを含む、各 4D Webサーバーは、`4D.WebServer` クラスの オブジェクトとして公開されます。インスタンス化された Webサーバーオブジェクトは、[多数のプロパティや関数](#) を使用して、カレントのアプリケーションまたは任意のコンポーネントから操作することができます。

4Dランゲージの従来の [WEBコマンド](#) はサポートされていますが、その対象となる Webサーバーを選択することはできません（後述参照）。

各 Webサーバー（ホストアプリケーションまたはコンポーネント）は、個別のコンテキストで使用できます。これには、以下が含まれます：

- `On Web Authentication` および `On Web Connection` データベースメソッドの呼び出し
- 4Dタグの処理とメソッドの呼び出し
- Webセッションや TLSプロトコルの管理

これにより、独自の Webインターフェースを備えた独立したコンポーネントや機能を開発することができます。

Webサーバーオブジェクトのインスタンス化

ホストアプリケーション（デフォルトWebサーバー）の Webサーバーオブジェクトは、4D 起動時に自動的に読み込まれます。したがって、新規作成したプロジェクトに次のように書いた場合：

```
$nbSrv:=WEB Server list.length  
// $nbSrv の値は 1
```

Webサーバーオブジェクトをインスタンス化するには、`WEB Server` コマンドを呼び出します。

```
// 4D.WebServer クラスのオブジェクト変数を作成します。  
var webServer : 4D.WebServer  
    // カレントコンテキストから Webサーバーを呼び出します  
webServer:=WEB Server  
  
    // 以下と同じです  
webServer:=WEB Server(Web server database)
```

アプリケーションがコンポーネントを使用している場合に：

- コンポーネントからホストアプリケーションの Webサーバーを呼び出す場合や
- リクエストを受け取ったサーバー（どのサーバーでも）を呼び出す場合

次をすることもできます：

```

var webServer : 4D.WebServer
    // コンポーネントからホストの Webサーバーを呼び出す
webServer:=WEB Server(Web server host database)
    // ターゲットの Webサーバーを呼び出す
webServer:=WEB Server(Web server receiving request)

```

Webサーバー関数

Webサーバークラスのオブジェクトには、以下の機能があります。

関数	引数	戻り値	説明
<code>start()</code>	<code>settings</code> (オブジェクト)	<code>status</code> (オブジェクト)	Webサーバーを開始します
<code>stop()</code>	-	-	Webサーバーを停止します

Webサーバーを起動・停止するには、Webサーバーオブジェクトの `start()` および `stop()` 関数を呼び出すだけです。

```

var $status : Object
    // デフォルトの設定で Webサーバーを起動する場合
$status:=webServer.start()
    // カスタム設定で Webサーバーを開始する場合
    // $settings オブジェクトは、Webサーバープロパティを格納します
webServer.start($settings)

    // Webサーバーを停止します
$status:=webServer.stop()

```

Webサーバープロパティ

Webサーバーオブジェクトには、Webサーバーを構成する [さまざまなプロパティ](#) が含まれています。

これらのプロパティは以下のように定義します:

1. `.start()` 関数の `settings` パラメーターを使用して定義します (読み取り専用のプロパティを除く、後述参照)。
2. 上を使用しない場合は、`WEB SET OPTION` コマンドを使用して定義します (ホストアプリケーションのみ)。
3. 上を使用しない場合は、ホストアプリケーションまたはコンポーネントの設定で定義します。
 - Webサーバーを起動していない場合、プロパティには Webサーバーの次回起動時に使用される値が含まれています。
 - Webサーバーが起動されている場合、プロパティには Webサーバーで使用される実際の値が含まれます (デフォルトの定義 `.start()` 関数の `settings` パラメーターによって上書きされている可能性があります)。

`isRunning`、`name`、`openSSLVersion`、`perfectForwardSecrecy` は読み取り専用のプロパティで、`start()` 関数の `settings` オブジェクトパラメーターで事前に定義することはできません。

4D Webコマンドのスコープ

4Dランゲージには、Webサーバーの制御に使用できる [いくつかのコマンド](#) があります。ただし、これらのコマンドは 1つの (デフォルト) Webサーバーで動作するように設計されています。これらのコマンドを Webサーバーオブジェクトのコンテキストで使用する場合は、そのスコープが適切であることを確認してください。

コマンド	スコープ
SET DATABASE PARAMETER	ホストアプリケーション Webサーバー
WEB CLOSE SESSION	リクエストを受け取った Webサーバー
WEB GET BODY PART	リクエストを受け取った Webサーバー
WEB Get body part count	リクエストを受け取った Webサーバー
WEB Get Current Session ID	リクエストを受け取った Webサーバー
WEB GET HTTP BODY	リクエストを受け取った Webサーバー
WEB GET HTTP HEADER	リクエストを受け取った Webサーバー
WEB GET OPTION	ホストアプリケーション Webサーバー
WEB Get server info	ホストアプリケーション Webサーバー
WEB GET SESSION EXPIRATION	リクエストを受け取った Webサーバー
WEB Get session process count	リクエストを受け取った Webサーバー
WEB GET STATISTICS	ホストアプリケーション Webサーバー
WEB GET VARIABLES	リクエストを受け取った Webサーバー
WEB Is secured connection	リクエストを受け取った Webサーバー
WEB Is server running	ホストアプリケーション Webサーバー
WEB SEND BLOB	リクエストを受け取った Webサーバー
WEB SEND FILE	リクエストを受け取った Webサーバー
WEB SEND HTTP REDIRECT	リクエストを受け取った Webサーバー
WEB SEND RAW DATA	リクエストを受け取った Webサーバー
WEB SEND TEXT	リクエストを受け取った Webサーバー
WEB SET HOME PAGE	ホストアプリケーション Webサーバー
WEB SET HTTP HEADER	リクエストを受け取った Webサーバー
WEB SET OPTION	ホストアプリケーション Webサーバー
WEB SET ROOT FOLDER	ホストアプリケーション Webサーバー
WEB START SERVER	ホストアプリケーション Webサーバー
WEB STOP SERVER	ホストアプリケーション Webサーバー
WEB Validate digest	リクエストを受け取った Webサーバー

テンプレートページ

4D の Web サーバーでは、タグを含む HTML テンプレートページを使用することができます。つまり、静的な HTML コードと、4DTEXT、4DIF、4DINCLUDEなどの[変換タグ](#)によって追加された 4D 参照の組み合わせです。これらのタグは通常、HTML タイプのコメント（`<!--#4DTagName TagValue-->`）として、HTML ソースコードに挿入されます。

これらのページが HTTP サーバーから送信される際、ページは解析され、含まれているタグが実行され、結果のデータに置き換えられます。このように、ブラウザが受け取るページは、静的な要素と 4D の処理による値が組み合ったものです。

たとえば、HTML ページ内にて以下のように記述すると：

```
<P><!--#4DTEXT vtSiteName--> へようこそ！</P>
```

4D 変数 *vtSiteName* の値が HTML ページに挿入されます。

テンプレート用タグ

以下の 4D タグを使用することができます：

- 4DTEXT: 4D 変数および式をテキストとして挿入します。
- 4DHML: HTML コードを挿入します。
- 4DEVAL: 4D 式を評価します。
- 4DSCRIPT: 4D メソッドを実行します。
- 4DINCLUDE: ページを他のページに含めます。
- 4DBASE: 4DINCLUDE タグが使用するデフォルト フォルダーを変更します。
- 4DCODE: 4D コードを挿入します。
- 4DIF, 4ELSE, 4ELSEIF, 4ENDIF: HTML コードに条件式を挿入します。
- 4DLOOP, 4ENDLOOP: HTML コードにループを挿入します。
- 4DEACH, 4ENDDEACH: コレクション内、エンティティセレクション内、またはオブジェクトのプロパティをループします。

これらのタグについては、[変換タグ](#) のページで説明しています。

タグは混在させることができます。たとえば、次のような HTML コードが認められています：

```
<HTML>
...
<BODY>
<!--#4DSCRIPT/PRE_PROCESS-->    (メソッド呼び出し)
<!--#4DIF (myvar=1)-->    (If 条件)
    <!--#4DINCLUDE banner1.html-->    (サブページ挿入)
<!--#4ENDIF-->    (End if)
<!--#4DIF (myvar=2)-->
    <!--#4DINCLUDE banner2.html-->
<!--#4ENDIF-->

<!--#4DLOOP [TABLE]-->    (カレントセレクションでのループ)
<!--#4DIF ([TABLE]ValNum>10)-->    (If [TABLE]ValNum>10)
    <!--#4DINCLUDE subpage.html-->    (サブページの挿入)
<!--#4ELSE-->    (Else)
    <B>Value: <!--#4DTEXT [TABLE]ValNum--></B><BR>    (フィールド表示)
<!--#4ENDIF-->
<!--#4DENDLOOP-->    (End for)
</BODY>
</HTML>
```

タグの解析

最適化のため、".HTML" または ".HTM" を末尾に持つ単純な URL で HTMLページを呼び出した場合、4D Webサーバーは HTMLソースコードの解析をおこないません。

4D Web サーバーが送信するテンプレートページの解析は、WEB SEND FILE (.htm, .html, .shtm, .shtml)、WEB SEND BLOB (text/html type BLOB)、または WEB SEND TEXT コマンドが呼び出されたとき、および URL を使用して呼び出されたページを送信するときにおこなわれます。URL で呼び出す場合、".htm" と ".html" で終わるページは最適化のため解析されません。この場合に HTMLページの解析を強制するには、末尾を ".shtm" または ".shtml" とする必要があります (例: <http://www.server.com/dir/page.shtm>)。このタイプのページの使用例は、WEB GET STATISTICS コマンドの説明に記載されています。XMLページ (.xml, .xsl) もサポートされており、常に 4D によって解析されます。

PROCESS 4D TAGS コマンドを使用すると、Webコンテキスト外でも解析をおこなうことができます。

内部的には、パーサーは UTF-16 文字列で動作しますが、解析するデータは異なる方法でエンコードされている場合があります。タグにテキストが含まれている場合 (4DHTML など)、提供されている情報 (charset) や出所に応じて、4D は必要に応じてデータを変換します。以下に、HTMLページに含まれるタグを 4D が解析する場合と、おこなわれる変換を示します：

動作 / コマンド	送信ページの解析	\$シンタックスのサポート(*)	タグ解析に使用される文字セット
URL で呼び出されたページ	<input checked="" type="radio"/> 、ただし拡張子が ".htm" または ".html" のページを除く	<input checked="" type="radio"/> 、ただし拡張子が ".htm" または ".html" のページを除く	ページの "Content-Type" ヘッダーのパラメーターに渡された文字セットに従います。それが無い場合は、META-HTTP EQUIVタグを探します。それも無ければ、HTTPサーバーのデフォルト文字セットを使用します。
WEB SEND FILE	<input checked="" type="radio"/>	-	ページの "Content-Type" ヘッダーのパラメーターに渡された文字セットに従います。それが無い場合は、META-HTTP EQUIVタグを探します。それも無ければ、HTTPサーバーのデフォルト文字セットを使用します。
WEB SEND TEXT	<input checked="" type="radio"/>	-	変換は不要です。
WEB SEND BLOB	<input checked="" type="radio"/> (BLOBが "text/html" 型の場合)	-	レスポンスの "Content-Type" ヘッダーに指定された文字セットを使います。それも無ければ、HTTPサーバーのデフォルト文字セットを使用します。
<!-- 4DINCLUDE --> タグによる挿入	<input checked="" type="radio"/>	<input checked="" type="radio"/>	ページの "Content-Type" ヘッダーのパラメーターに渡された文字セットに従います。それが無い場合は、META-HTTP EQUIVタグを探します。それも無ければ、HTTPサーバーのデフォルト文字セットを使用します。
PROCESS 4D TAGS	<input checked="" type="radio"/>	<input checked="" type="radio"/>	テキストデータは変換しません。BLOBデータは、互換性のために Mac-Roman 文字セットから自動変換されます。

(*) 4DHTML、4DTEXT、4DEVALタグにおいては、代替の \$ベースシンタックスが利用可能です。

Webから 4Dメソッドへのアクセス

4DEACH、4DELSEIF、4DEVAL、4DHTML、4DIF、4DLOOP、4DSRIPT、または 4DTEXT で Webリクエストから 4Dメソッドを実行できるか否かは、メソッドプロパティの "公開オプション: 4D タグと URL(4DACTION...)" 属性の設定に依存します。この属性がチェックされていないメソッドは、Webリクエストから呼び出すことができません。

悪意あるコードの侵入を防止

4D変換タグは様々なタイプのデータを引数として受け入れます：テキスト、変数、メソッド、コマンド名、…。これらのデータが自分で書いたコードから提供される場合、その受け渡しを自分でコントロールできるので、悪意あるコードの侵入のリスクは無いと言つていいでしょう。しかしながら、データベースのコード扱うデータは多くの場合、外部ソース（ユーザー入力、読み込み、等）から導入されたものです。

この場合、4DEVAL や 4DSRIPT などの変換タグは 使用しないのが賢明です。なぜならこれらのタグはこういったデータが格納された変数を直接評価するからです。

これに加え、繰り返しの原則 に従い、悪意あるコード自身が変換タグを含んでいる可能性もあります。この場合、4DTEXT タグを使用しなくてはなりません。

せん。例として、"Name" という名前の Web フォームフィールドがあり、ユーザーがそこに名前を入力する場合を考えます。この名前は <!--#4DHTML vName--> タグを使用してページ内に表示されます。もし " <!--#4DEVAL QUIT 4D--> " というテキストが名前の代わりに入力されたとしたら、このタグを解釈するとアプリケーションは終了てしまいます。このリスクを避けるには、4DTEXT タグを使用することで対応できます。このタグは特殊 HTML 文字をエスケープするため、挿入された悪意ある再起的コードが解釈されることはありません。前の例でいうと、"Name" フィールドには " <!--#4DEVAL QUIT 4D--> " が含まれることになり、これは変換されません。

HTTPリクエストの処理

4D Webサーバーは、HTTPリクエストを処理するための機能を複数備えています：

- Webアプリケーションのルーターとなる `On Web Connection` データベースメソッド。
- サーバーサイドコードを呼び出すための `/4DACTION` URL。
- サーバーに送信された HTMLオブジェクトから値を取得する `WEB GET VARIABLES`。
- `WEB GET HTTP BODY`、`WEB GET HTTP HEADER`、`WEB GET BODY PART` などのコマンドによって、リクエスト処理をカスタマイズすることができます (cookie 含む)。
- 変数を宣言するための `COMPILER_WEB` プロジェクトメソッド。

On Web Connection

`On Web Connection` データベースメソッドは、4D Webサーバーのエントリーポイントとして使用できます。

データベースメソッドの呼び出し

`On Web Connection` データベースメソッドは、サーバー上に存在しないページへのパスをサーバーが URL として受け取った場合に、自動的に呼び出されます。データベースメソッドは、URL とともに呼び出されます。

たとえば、"a/b/c" という URL はデータベースメソッドを呼び出しますが、[WebFolder](#) の "a/b" サブフォルダーに "c.html" というページが存在する場合、"a/b/c.html" はデータベースメソッドを呼び出しません。

このリクエストは、事前に `On Web Authentication` データベースメソッド (あれば) に受け入れられているべきで、Webサーバーも起動している必要があります。

シンタックス

`On Web Connection($1 : Text ; $2 : Text ; $3 : Text ; $4 : Text ; $5 : Text ; $6 : Text)`

引数	タイプ		説明
\$1	テキスト	<-	URL
\$2	テキスト	<-	HTTPヘッダー + HTTPボディ (32 KBまで)
\$3	テキスト	<-	Webクライアント (ブラウザー) の IPアドレス
\$4	テキスト	<-	サーバーの IPアドレス
\$5	テキスト	<-	ユーザー名
\$6	テキスト	<-	パスワード

これらの引数を以下のように宣言しなければなりません：

```
// On Web Connection データベースメソッド  
  
C_TEXT($1;$2;$3;$4;$5;$6)  
  
// メソッドのコード
```

あるいは、[名前付き引数](#) シンタックスを利用することもできます：

```
// On Web Connection データベースメソッド  
#DECLARE ($url : Text; $header : Text; \  
$BrowserIP : Text; $ServerIP : Text; \  
$user : Text; $password : Text)
```

インターフェース要素 を表示する 4Dコマンド (DIALOG 、 ALERT など) の呼び出しは許可されず、メソッドの処理を終了します。

\$1 - URL追加データ

最初の引数 (\$1) は、ユーザーが Webブラウザーのアドレスエリアに入力した URL からホストのアドレスを取り除いたものです。

インターネット接続の場合を見てみましょう。4D Webサーバーマシンの IPアドレスを 123.4.567.89 とします。以下の表は Webブラウザーに入力された URL に対して、\$1 が受け取る値を示しています:

Webブラウザーに入力された値	\$1 の値
123.4.567.89	/
http://123.4.567.89	/
123.4.567.89/Customers	/Customers
http://123.4.567.89/Customers/Add	/Customers/Add
123.4.567.89/Do_This/If_OK/Do_That	/Do_This/If_OK/Do_That

この引数は必要に応じて自由に利用できます。4D は単に URL のホスト部より後の部分を無視し、\$1 に渡します。たとえば、値 "/Customers/Add" が " [Customers] テーブルに新規レコードを直接追加する" ということを意味するような、オリジナルのルールを作成できます。利用可能な値やデフォルトブックマークを Webユーザーに提供することで、アプリケーションの異なる部分へのショートカットを提供できます。このようにして、Webユーザーは新しく接続するたびにナビゲーションを通過することなく、素早く Webサイトのリソースにアクセスできます。

\$2 - HTTPリクエストのヘッダーとボディ

二番目の引数 (\$2) は、Webブラウザーから送信された HTTPリクエストのヘッダーとボディです。この情報は On Web Connection データベースメソッドに "そのまま" 渡されることに留意してください。その内容は、接続を試みた Webブラウザーの仕様により異なります。

アプリケーションでこの情報を使用するには、開発者がヘッダーとボディを解析しなければなりません。 WEB GET HTTP HEADER や WEB GET HTTP BODY コマンドを使うことができます。

パフォーマンス上の理由により、\$2 を介して渡されるデータのサイズは 32KB 以下でなくてはなりません。これを超過する分は、4D HTTPサーバーにより切り取られます。

\$3 - Webクライアントの IPアドレス

\$3 引数はブラウザーマシンの IPアドレスを受け取ります。この情報を使用して、インターネットアクセスとインターネットアクセスを区別できます。

4D は IPv4 アドレスを、96-bit の接頭辞付きのハイブリッド型 IPv6/IPv4 フォーマットで返します。たとえば、::ffff:192.168.2.34 は、192.168.2.34 という IPv4 アドレスを意味します。詳細については、[IPv6 のサポートについて](#) の章を参照ください。

\$4 - サーバー IPアドレス

\$4 引数は 4D Webサーバーによってリクエストされた IPアドレスを受け取ります。4D はマルチホーミングをサポートしており、複数の IPアドレスを持つマシンを使用できます。詳細は [設定ページ](#) を参照ください。

\$5 と \$6 - ユーザー名とパスワード

\$5 と \$6 引数は、ブラウザーが表示する標準の認証ダイアログにユーザーが入力したユーザー名とパスワードを受け取ります (入力されていれば；認

[証ページ 参照\)。](#)

ブラウザから送信されたユーザー名が 4D に存在する場合、\$6 引数 (ユーザー名/パスワード) はセキュリティのため渡されません。

/4DACTION

/4DACTION/MethodName***
*/4DACTION/*****MethodName/Param

引数	タイプ		説明
MethodName	テキスト	->	実行する 4D プロジェクトメソッド名
Param	テキスト	->	プロジェクトメソッドに渡されるテキスト引数

利用法: URL またはフォームアクション

この URL を使用して、任意の *Param* テキスト引数とともに *MethodName* に指定した 4D プロジェクトメソッドを呼び出すことができます。このメソッドは引数を \$1 に受け取ります。

- 4D プロジェクトメソッドは、[Webリクエスト用に許可](#)されていないなりません。メソッドのロパティで "公開オプション: 4Dタグと URL(4DACTION...)" 属性がチェックされている必要があります。属性がチェックされていない場合、Webリクエストは拒否されます。
- `/4DACTION/MyMethod/Param` リクエストを受け取ると、4D は `On Web Authentication` データベースメソッド (あれば) を呼び出します。

4DACTION/ は、スタティックな Web ページの URL に割り当てることもできます:

```
<A HREF="/4DACTION/MyMethod/hello">Do Something</A>
```

MyMethod プロジェクトメソッドは通常レスポンスを返すべきです (`WEB SEND FILE` や `WEB SEND BLOB` で HTML ページを送信するなど)。ブラウザをブロックしないように、処理は可能な限り短時間でおこなわれるようになります。

4DACTION/ から呼び出されるメソッドは、インターフェース要素 (`DIALOG`, `ALERT` など) を呼び出してはいけません。

例題

この例題は、HTML ピクチャーオブジェクトに `/4DACTION/` URL を割り当て、ページ上でピクチャーを動的に表示する方法を説明しています。スタティック HTML ページに以下のコードを記述します:

```
<IMG SRC="/4DACTION/getPhoto/smith">
```

`getPhoto` メソッドは以下のとおりです:

```
C_TEXT($1) // この引数は常に宣言する必要があります
var $path : Text
var $PictVar : Picture
var $BlobVar : Blob

// Resources フォルダー内の Images フォルダー内でピクチャーを探します
$path:=Get 4D folder(Current resources folder)+"Images"+Folder separator+$1+".psd"

READ PICTURE FILE($path;$PictVar) // ピクチャーをピクチャー変数に入れます
PICTURE TO BLOB($PictVar;$Blob;"png") // ピクチャーを ".png" 形式に変換します
WEB SEND BLOB($Blob;"image/png")
```

4DACTION を使用してフォームをポスト

4D Webサーバーでは、ポストされたフォームを使用することもできます。これはスタティックなページから Webサーバーにデータを送信し、すべての値を簡単に取得するというものです。POSTタイプを使用し、フォームのアクションは /4DACTION/MethodName で始まつていなければなりません。

フォームは 2つのメソッドを使用してサブミットできます (4D では両方のタイプを使用できます):

- POST は通常 Webサーバーにデータを送信するのに使用します。
- GET は通常 Webサーバーからデータをリクエストするのに使用します。

この場合、Webサーバーがポストされたフォームを受信すると、On Web Authentication データベースメソッドが (あれば) 呼び出されます。

サーバーに投稿された HTMLページに含まれるすべてのフィールドの [名前と値を取得](#) するには、このメソッド内で WEB GET VARIABLES コマンドを呼び出す必要があります。

フォームのアクションを定義する例:

```
<FORM ACTION="/4DACTION/MethodName" METHOD=POST>
```

例題

Webアプリケーションにおいて、スタティックなHTMLページを使い、ブラウザからレコードを検索できるようにしたいとします。このページを "search.htm" とします。アプリケーションには、検索結果を表示するためのスタティックページ ("results.htm") もあるとします。POSTメソッドと /4DACTION/PROCESSFORM アクションがページに割り当てられています。

以下はこのページの HTMLコードです:

```
<form action="/4daction/processForm" method=POST>
<input type=text name=vName value=""><BR>
<input type=checkbox name=vExact value="Word">Whole word<BR>
<input type=submit name=OK value="Search">
</FORM>
```

データ入力エリアに "ABCD" とタイプし、"Whole word (句として検索)" オプションをチェックして Search (検索) ボタンをクリックします。Webサーバーに送信されるリクエスト内部は以下の通りです:

```
vName="ABCD"
vExact="Word"
OK="Search"
```

4D は On Web Authentication データベースメソッドを (あれば) 呼び出し、そして以下の processForm プロジェクトメソッドを呼び出します:

```

C_TEXT($1) // コンパイルモードの場合必須
C_LONGINT($vName)
C_TEXT(vName;vLIST)
ARRAY TEXT($arrNames;0)
ARRAY TEXT($arrVals;0)
WEB GET VARIABLES($arrNames;$arrVals) // フォーム上の変数をすべて取得します
$vName:=Find in array($arrNames;"vName")
vName:=$arrVals{$vName}
If(Find in array($arrNames;"vExact")=-1) // オプションがチェックされていない場合
    vName:=vName+"@"
End if
QUERY([Jockeys];[Jockeys]Name=vName)
FIRST RECORD([Jockeys])
While(Not(End selection([Jockeys])))
    vLIST:=vLIST+[Jockeys]Name+" "+[Jockeys]Tel+"<BR>"
    NEXT RECORD([Jockeys])
End while
WEB SEND FILE("results.htm") // 検索結果が挿入される results.htm を送信します
// このページには変数 vLIST の参照が含まれています
// たとえば <!--4DHTML vLIST--> など
// ...
End if

```

HTTPリクエストから値を取得する

4D Web サーバーでは、Webフォームや URL を介して POST や GET リクエストで送信されたデータを復元することができます。

ヘッダーや URL にデータが含まれたリクエストを Webサーバーが受信すると、4D はそれに含まれる HTMLオブジェクトの値を受け取ることができます。たとえば `WEB SEND FILE` コマンドまたは `WEB SEND BLOB` コマンドで送信され、ユーザーが値を入力・修正して確定ボタンをクリックするような Web フォームにおいてもこの原理は使用可能です。

この場合 4D は `WEB GET VARIABLES` コマンドを使って、リクエスト内の HTMLオブジェクトの値を取得することができます。`WEB GET VARIABLES` コマンドは、値をテキストとして受け取ります。

以下の HTMLページのソースコードがあるとき：

```

<html>
<head>
    <title>Welcome</title>
    <script language="JavaScript"><!--
function GetBrowserInformation(formObj){
formObj.vtNav_appName.value = navigator.appName
formObj.vtNav_appVersion.value = navigator.appVersion
formObj.vtNav_appCodeName.value = navigator.appCodeName
formObj.vtNav_userAgent.value = navigator.userAgent
return true
}
function LogOn(formObj){
if(formObj.vtUserName.value!=""){
return true
} else {
alert("Enter your name, then try again.")
return false
}
}
//--></script>
</head>
<body>
<form action="/4DACTION/WWW_STD_FORM_POST" method="post"
name="frmWelcome"
onsubmit="return GetBrowserInformation(frmWelcome)">
    <h1>Welcome to Spiders United</h1>
    <p><b>Please enter your name:</b>
    <input name="vtUserName" value="" size="30" type="text"></p>
    <p>
    <input name="vsbLogOn" value="Log On" onclick="return LogOn(frmWelcome)" type="submit">
    <input name="vsbRegister" value="Register" type="submit">
    <input name="vsbInformation" value="Information" type="submit"></p>
    <p>
    <input name="vtNav_appName" value="" type="hidden">
    <input name="vtNav_appVersion" value="" type="hidden">
    <input name="vtNav_appCodeName" value="" type="hidden">
    <input name="vtNav_userAgent" value="" type="hidden"></p>
</form>
</body>
</html>

```

4D が Web ブラウザーにページを送信すると、以下のように表示されます：

Welcome to Spiders United

Please enter your name:

このページの主な特徴は：

- 送信のための Submit ボタンが 3つあります： `vsbLogOn` , `vsbRegister` そして `vsbInformation` 。
- `Log On` をクリックすると、フォームからの送信はまず初めに JavaScript 関数 `LogOn` によって処理されます。名前が入力されていない場合、フォームは 4D に送信すらされず、JavaScript による警告が表示されます。
- フォームは POST 4D メソッドに加えて、ブラウザープロパティを `vtNav_App` から始まる名称の 4 つの隠しオブジェクトへとコピーする投稿スクリプト (`GetBrowserInformation`) を持っています。また、このページには `vtUserName` オブジェクトも含まれます。

ユーザーが HTML フォーム上のボタンのどれかをクリックした際に呼び出される `WWW_STD_FORM_POST` という 4D メソッドを検証してみましょう。

```

// 変数の値を取得します
ARRAY TEXT($arrNames;0)
ARRAY TEXT($arrValues;0)
WEB GET VARIABLES($arrNames;$arrValues)
C_TEXT($user)

Case of

// Log On ボタンがクリックされた場合
:(Find in array($arrNames;"vsbLogOn")#-1)
$user :=Find in array($arrNames;"vtUserName")
QUERY([WWW Users];[WWW Users]UserName=$arrValues{$user})
$0:=(Records in selection([WWW Users])>0)
If($0)
    WWW POST EVENT("Log On";WWW Log information)
// WWW POST EVENT メソッドが情報をデータベースのテーブルに保存します
Else

    $0:=WWW Register
// WWW Register メソッドは新規 Webユーザーの登録を処理します
End if

// Register ボタンがクリックされた場合
:(Find in array($arrNames;"vsbRegister")#-1)
$0:=WWW Register

// Information ボタンがクリックされた場合
:(Find in array($arrNames;"vsbInformation")#-1)
    WEB SEND FILE("userinfos.html")
End case

```

このメソッドの機能は:

- 変数 *vtNav_appName*, *vtNav_appVersion*, *vtNav_appCodeName*, そして *vtNav_userAgent* の値 (同じ名前を持つ HTMLオブジェクトにそれぞれバインドされています) は、WEB GET VARIABLES コマンドを使用することによって JavaScript のスクリプト *GetBrowserInformation* で作成された HTMLオブジェクトから取得することができます。
- 3つの投稿ボタンにバインドされている変数 *vsbLogOn*, *vsbRegister* と *vsbInformation* のうち、クリックされたボタンに対応するもののみが WEB GET VARIABLES コマンドによって取得されます。この 3つのうちいずれかのボタンによって投稿がおこなわれたとき、ブラウザーはクリックされたボタンの値を 4D に返します。これにより、どのボタンがクリックされたのかが分かります。

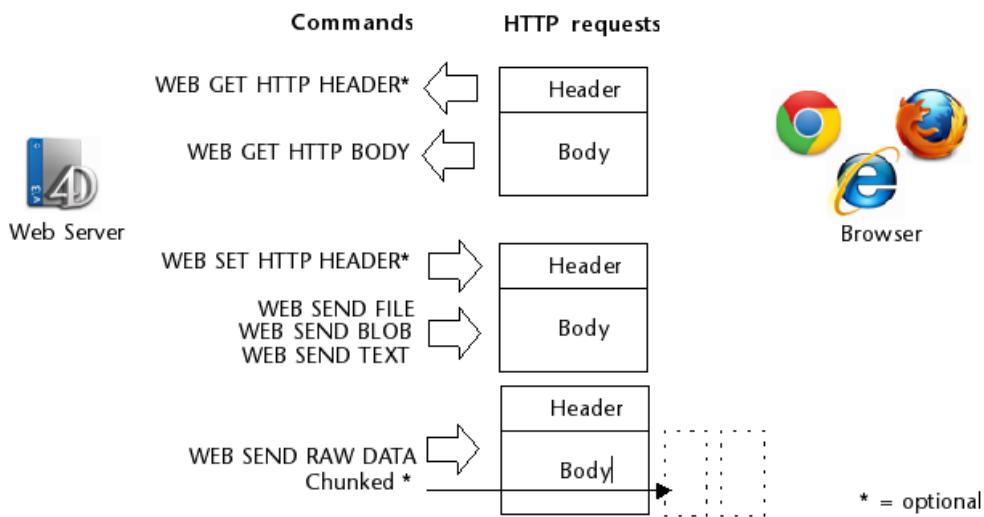
HTMLではすべてのオブジェクトがテキストオブジェクトであることに留意が必要です。SELECT要素を使用した場合、WEB GET VARIABLES コマンドで返されるのはオブジェクト内でハイライトされている要素の値であり、4D のように配列内の要素の位置を返すわけではありません。WEB GET VARIABLES コマンドは必ずテキスト型の値を返します。

その他の Webサーバーコマンド

4D Webサーバーには、リクエストの処理をカスタマイズするための、低レベル Webコマンドがいくつか用意されています。

- WEB GET HTTP BODY コマンドは、ボディをそのままの状態でテキストとして返します。これを必要に応じて解析することができます。
- WEB GET HTTP HEADER コマンドは、リクエストのヘッダーを返します。カスタムcookieなどを処理するのに便利です (WEB SET HTTP HEADER コマンドも使用できます)。
- WEB GET BODY PART と WEB Get body part count コマンドは、マルチパートリクエストのボディパートを解析して、テキスト値を取得するだけでなく、ポストされたファイルもBLOBに取得します。

これらのコマンドは次の図にまとめられています:



4D Webサーバーは、どの Webクライアントからでもチャンクド・エンコーディングでアップロードされたファイルをサポートするようになりました。チャンクド・エンコーディングは HTTP/1.1 にて定義されているデータ転送方式です。これを使用することにより、最終的なデータサイズを知る事なく、データを複数の "チャック" (部分) に分けて転送することができます。4D Webサーバーでは、サーバーから Webクライアントへのチャンクド・エンコーディングもサポートしています (WEB SEND RAW DATA を使用します)。

COMPILER_WEB プロジェクトメソッド

COMPILER_WEB メソッドが存在する場合、それは HTTPサーバーが動的なリクエストを受け取り、4Dエンジンを呼び出した場合に、システムを通して呼び出されます。これはたとえば 4D Webサーバーが、ポストされたフォーム、または処理すべき URL を [<On Web Connection](#) に受け取る場合が該当します。このメソッドは Web通信時に使用される型指定または変数初期化指示子を含めることを目的としています。これはデータベースのコンパイル時にコンパイラによって使用されます。COMPILER_WEB メソッドはすべての Webフォームで共通です。デフォルトでは、COMPILER_WEB メソッドは存在しません。明示的に作成する必要があります。

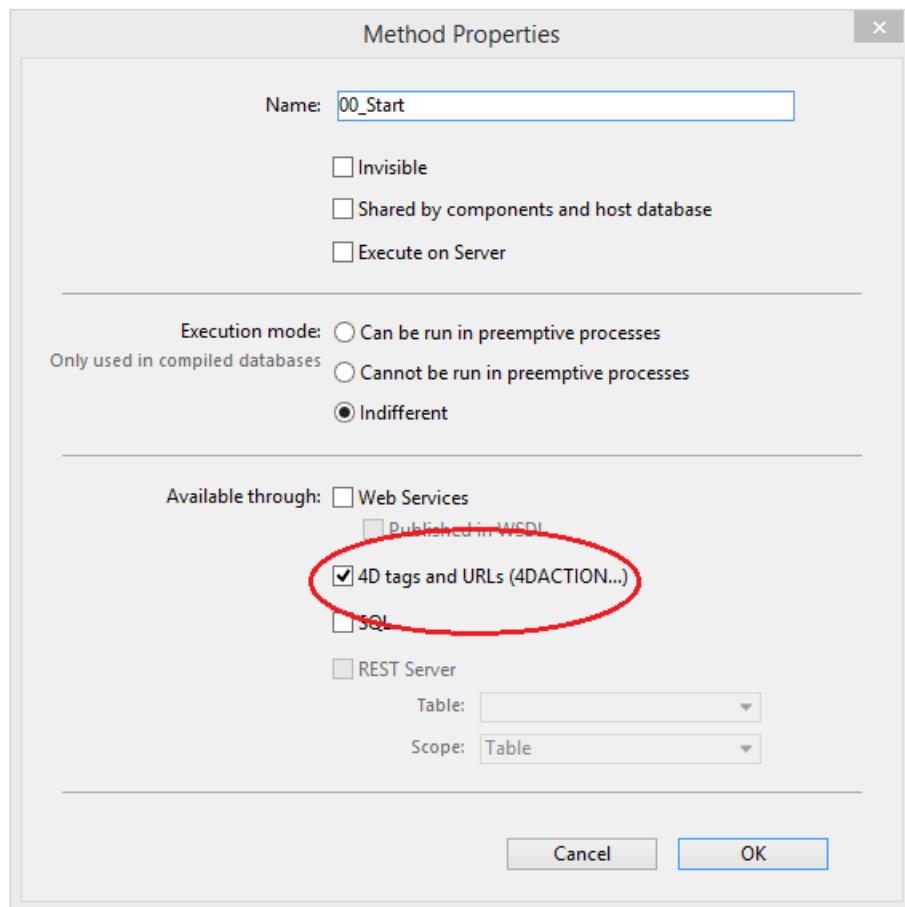
COMPILER_WEB プロジェクトメソッドは (存在すれば)、SOAPリクエストが受け入れられることに実行されます。

プロジェクトメソッドの許可

`4DEVAL`, `4DTEXT`, `4DHTML` などの 4Dタグや `/4ACTION URL` を使用すると、Webに公開された 4Dプロジェクトのあらゆるプロジェクトメソッドが実行できます。たとえば、リクエスト <http://www.server.com/4ACTION/login> は `login` プロジェクトメソッドを (存在すれば) 実行します。

このメカニズムは具体的には、インターネット上のユーザーが故意に (あるいは予期せず) Web用でないメソッドを実行してしまうというような、アプリケーションのセキュリティを脅かすリスクをもたらします。このリスクは以下の 3つの方法で回避できます:

- `On Web Authentication` データベースメソッドを使用して URL から呼び出されるメソッドをフィルターする。欠点: データベースに多くのメソッドが定義されている場合、この方法は管理が困難になります。
- 公開オプション: 4DタグとURL (4ACTION...) (メソッドプロパティ) を使用する:



このオプションを使用してプロジェクトメソッドごとに、特別な URL `4ACTION` や `4DTEXT`, `4DHTML`, `4DEVAL`, `4SCRIPT`, `4DIF`, `4DELSEIF`, `4DLOOP` などの 4Dタグを使用した呼び出しを許可するかしないかを設定できます。このオプションがチェックされていない場合、そのプロジェクトメソッドは HTTPリクエストから直接実行することはできません。他方、他のタイプの呼び出し (フォーミュラや他のメソッドからの呼び出しなど) ではこれらのメソッドを実行することができます。

このオプションはデフォルトでチェックされていません。`4ACTION` やタグなどを使用して呼び出すことのできるメソッドは、明示的に指定する必要があります。

このプロパティが指定されたプロジェクトメソッドは、エクスプローラーで以下のアイコンが表示されます:



カスタム HTTPエラーページ

4D Web Server を使って、サーバレスポンスのステータスコードに基づいて、クライアントに送信される HTTPエラーページをカスタマイズすることができます。エラーページとは、以下のものを指します：

- 4 から始まるステータスコード（クライアントエラー）。たとえば 404 など。
- 5 から始まるステータスコード（サーバーエラー）。たとえば 501 など。

HTTPエラーステータスコードの完全な詳細については、[HTTPステータスコード](#) (Wikipedia) を参照してください。

デフォルトページの置換

デフォルトの 4D Web Server エラーページを独自のページで置き換えるためには、以下のようにします：

- カスタム HTMLページをアプリケーションの WebFolder フォルダーの第1レベルに置きます。
- カスタムページを "{statusCode}.html" (例: "404.html") という名前にします。

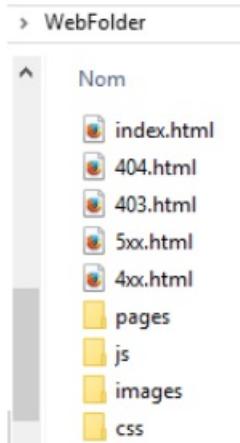
一つのステータスコードにつき、一つのエラーページを定義することができるほか、"{number}xx.html" と名前をつけることで複数のエラーに汎用的なエラーページを定義することもできます。たとえば、クライアントエラー全般に対するページとして、"4xx.html" というファイルを作成できます。4D Web Server は最初に {statusCode}.html のページを探し、それが存在しない場合には汎用的なページを探します。

たとえば、HTTPレスポンスがステータスコード 404 を返す場合：

1. 4D Web Server は、アプリケーションの WebFolder フォルダー内にある "404.html" ページを送信しようとします。
2. それが見つからない場合、4D Web Server はアプリケーションの WebFolder フォルダー内にある "4xx.html" ページを送信しようとします。
3. それも見つからない場合、4D Web Server はデフォルトのエラーページを使用します。

例題

WebFolder フォルダーに、以下のようにカスタムページを定義している場合：



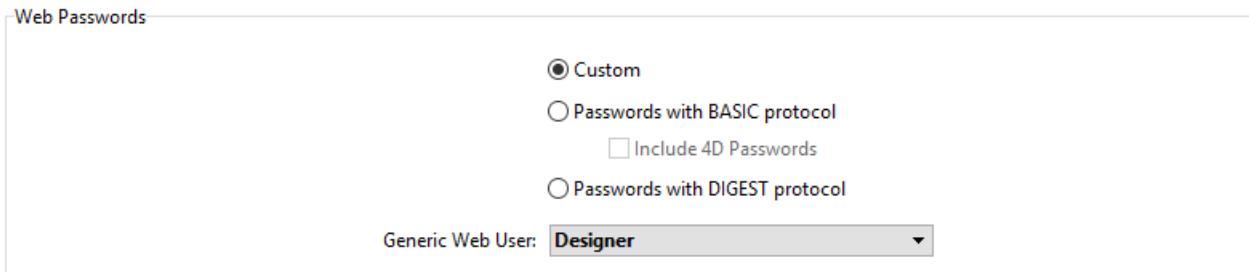
- 403、404 HTTPレスポンスに対しては、"403.html" および "404.html" ページがそれぞれ返されます。
- 他の 4xx エラーステータス (400、401など) に対しては、"4xx.html" ページが返されます。
- 5xx エラーステータス全般に対しては"5xx.html" ページが返されます。

認証

Webユーザーに特定のアクセス権を与えるには、ユーザーを認証することが必要です。認証とは、ユーザーの資格情報（通常は名前とパスワード）を取得・処理する方法のことです。

認証モード

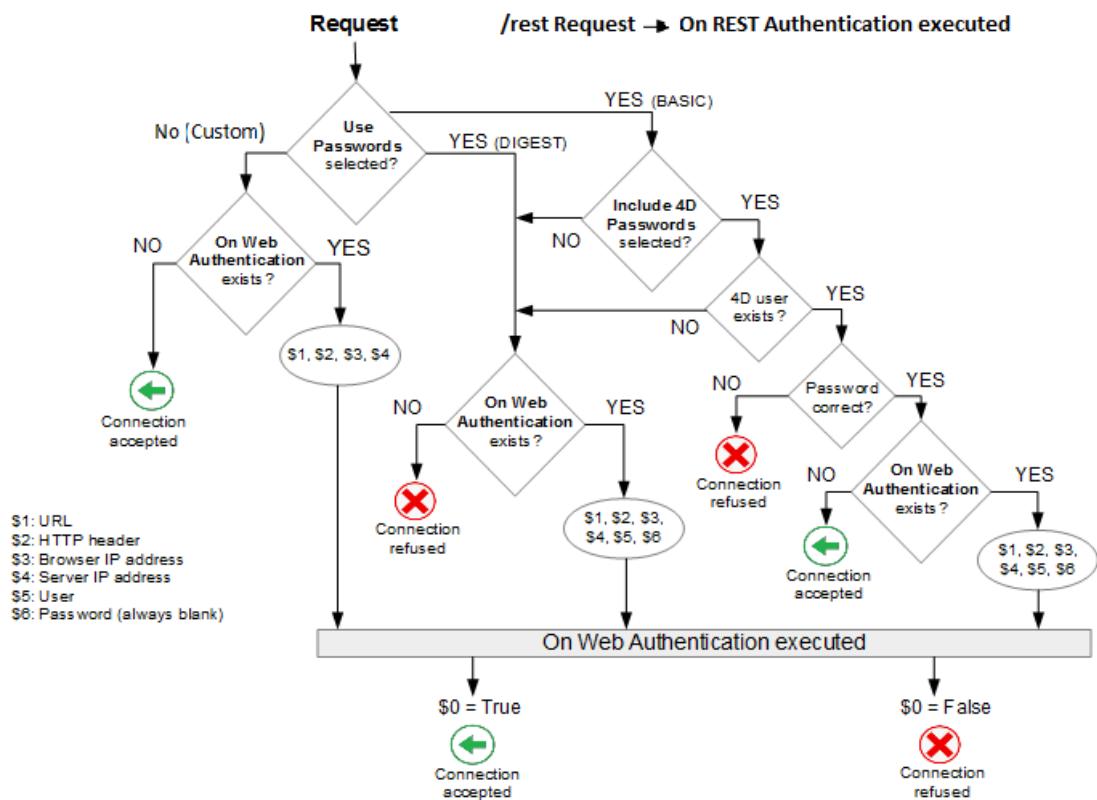
4D Webサーバーでは、3つの認証モードが用意されており、ストラクチャー設定ダイアログボックスの「Web/オプション (I)」ページで選択することができます：



カスタムの認証を使用することが推奨されています。

概要

4D Webサーバーのアクセスシステムの処理を以下に図示します：



rest/ で始まるリクエストは、RESTサーバーが直接処理します。

カスタムの認証（デフォルト）

このモードでは基本的に、ユーザーを認証する方法は開発者に委ねられています。4Dは、認証を必要とするHTTPリクエストのみを評価します。

この認証モードは最も柔軟性が高く、以下のことが可能です：

- ユーザー認証をサードパーティ・アプリケーション（例：ソーシャルネットワーク、SSO）に委ねることができます。
- 顧客データベースにアカウントを作成できるよう、ユーザーにインターフェース（Webフォームなど）を提供することもできます。登録後は、任意のカスタムアルゴリズムでユーザーを認証することができます（「ユーザー認証」の章の[例題](#)を参照ください）。重要なのは、以下のようなコードを使って、決してパスワードを平文で保存しないことです：

```
//... ユーザーアカウントの作成  
ds.webUser.password:=Generate password hash($password)  
ds.webUser.save()
```

"はじめに" の章の[例題](#)も参照ください。

カスタム認証が提供されていない場合、4D は [On Web Authentication](#) データベースメソッドを呼び出します（あれば）。\$1 と \$2 に加えて、ブラウザーとサーバーの IP アドレス（\$3 と \$4）のみが提供され、ユーザー名とパスワード（\$5 と \$6）は空です。ユーザー認証が成功した場合、このメソッドは \$0 に True を返さなければなりません。この場合、リクエストされたリソースが提供されます。認証が失敗した場合には、\$0 に False を返します。

警告：[On Web Authentication](#) データベースメソッドが存在しない場合、接続は自動的に受け入れられます（テストモード）。

BASIC認証

ユーザーがサーバーに接続するとダイアログボックスがブラウザー上に表示され、ユーザー名とパスワードの入力を求められます。

ユーザーが入力したユーザー名とパスワードは暗号化されずに、HTTPリクエストヘッダーに含められて送信されます。このモードで機密性を確保するには通常、HTTPSを必要とします。

入力された値は次のように評価されます：

- 4Dパスワードを含むオプションがチェックされている場合、ユーザーの認証情報はまず、[内部の4Dユーザーテーブル](#)に対して評価されます。
 - ブラウザーから送信されたユーザー名が 4D のユーザーテーブルに存在し、パスワードが正しい場合、接続は受け入れられます。パスワードが正しくなければ接続は拒否されます。
 - ユーザー名が 4D のユーザーテーブルに存在しない場合、[On Web Authentication](#) データベースメソッドが呼び出されます。[On Web Authentication](#) データベースメソッドが存在しない場合、接続は拒否されます。
- 4Dパスワードを含むオプションがチェックされていない場合、ユーザーの認証情報は、その他の接続情報（IPアドレス、ポート、URL など）とともに [On Web Authentication](#) データベースメソッドに受け渡されます。[On Web Authentication](#) データベースメソッドが存在しない場合、接続は拒否されます。

4Dクライアントの Webサーバーでは、すべての 4Dクライアントマシンが同じユーザーテーブルを共有することに留意が必要です。ユーザー名/パスワードの検証は 4D Serverアプリケーションでおこなわれます。

DIGEST認証

DIGESTモードはより高いセキュリティレベルを提供します。認証情報は復号が困難な一方向ハッシュを使用して処理されます。

BASICモードと同様に、ユーザーは接続時に自分の名前とパスワードを入力する必要があります。その後、[On Web Authentication](#) データベースメソッドが呼び出されます。DIGESTモードが有効の時、\$6引数（パスワード）は常に空の文字列が渡されます。実際このモードを使用するとき、この情報はネットワークからクリアテキスト（平文）では渡されません。この場合、接続リクエストは [WEB Validate digest](#) コマンドを使用して検証しなければなりません。

これらのパラメーターの変更を反映させるためには、Webサーバーを再起動する必要があります。

On Web Authentication

[On Web Authentication](#) データベースメソッドは Webサーバーエンジンへのアクセス管理を担当します。4D または 4D Server は、動的な

HTTPリクエストを受け取ると、このデータベースメソッドを呼び出します。

データベースメソッドの呼び出し

`On Web Authentication` データベースメソッドは、リクエストや処理が 4Dコードの実行を必要とするとき (REST呼び出しを除く) に自動で呼び出されます。また、Webサーバーが無効な静的URLを受信した場合 (要求された静的ページが存在しない場合など) にも呼び出されます。

つまり、`On Web Authentication` データベースメソッドは次の場合に呼び出されます：

- Webサーバーが、存在しないリソースを要求する URL を受信した場合
- Webサーバーが `4DACTION/`, `4DCGI/` ... で始まる URL を受信した場合
- Webサーバーがルートアクセス URL を受信したが、ストラクチャー設定または `WEB SET HOME PAGE` コマンドでホームページが設定されていないとき
- Webサーバーが、セミダイナミックページ内でコードを実行するタグ (`4DSCRIPT` など) を処理した場合。

次の場合には、`On Web Authentication` データベースメソッドは呼び出されません：

- Webサーバーが有効な静的ページを要求する URL を受信したとき。
- Webサーバーが `rest/` で始まる URL を受信し、RESTサーバーが起動したとき (この場合、認証は `On REST Authentication` データベースメソッド または [ストラクチャー設定](#) によって処理されます)。

シンタックス

`On Web Authentication($1 : Text ; $2 : Text ; $3 : Text ; $4 : Text ; $5 : Text ; $6 : Text) -> $0 : Boolean`

引数	タイプ		説明
\$1	テキスト	<-	URL
\$2	テキスト	<-	HTTPヘッダー + HTTPボディ (32 KBまで)
\$3	テキスト	<-	Webクライアント (ブラウザー) の IPアドレス
\$4	テキスト	<-	サーバーの IPアドレス
\$5	テキスト	<-	ユーザー名
\$6	テキスト	<-	パスワード
\$0	ブール	->	True = リクエストは受け入れられました、False = リクエストが拒否されました

これらの引数を以下のように宣言しなければなりません：

```
// On Web Authentication データベースメソッド

C_TEXT($1;$2;$3;$4;$5;$6)
C_BOOLEAN($0)

// メソッドのコード
```

あるいは、[名前付き引数](#) シンタックスを利用することもできます：

```
// On Web Authentication データベースメソッド
#DECLARE ($url : Text; $header : Text; \
$BrowserIP : Text; $ServerIP : Text; \
$user : Text; $password : Text) \
-> $RequestAccepted : Boolean
```

`On Web Authentication` データベースメソッドのすべての引数が必ず値を受け取るわけではありません。データベースメソッドが受け取る情報は、[認証モード](#)の設定により異なります。

\$1 - URL

最初の引数 (\$1) は、ユーザーが Web ブラウザーのアドレスエリアに入力した URL からホストのアドレスを取り除いたものです。

インターネット接続の場合をみてみましょう。4D Web サーバーマシンの IP アドレスを 123.45.67.89 とします。以下の表は Web ブラウザーに入力された URL に対して、\$1 が受け取る値を示しています：

Web ブラウザーに入力された値	\$1 の値
123.45.67.89	/
http://123.45.67.89	/
123.45.67.89/Customers	/Customers
http://123.45.67.89/Customers/Add	/Customers/Add
123.45.67.89/Do_This/If_OK/Do_That	/Do_This/If_OK/Do_That

\$2 - HTTPリクエストのヘッダーとボディ

二番目の引数 (\$2) は、Web ブラウザーから送信された HTTP リクエストのヘッダーとボディです。この情報は `On Web Authentication` データベースメソッドに "そのまま" 渡されることに留意してください。その内容は、接続を試みた Web ブラウザーの仕様により異なります。

アプリケーションでこの情報を使用するには、開発者がヘッダーとボディを解析しなければなりません。`WEB GET HTTP HEADER` や `WEB GET HTTP BODY` コマンドを使うことができます。

パフォーマンス上の理由により、\$2 を介して渡されるデータのサイズは 32KB 以下でなくてはなりません。これを超過する分は、4D HTTP サーバーにより切り取られます。

\$3 - Web クライアントの IP アドレス

\$3 引数はブラウザーマシンの IP アドレスを受け取ります。この情報を使用して、インターネットアクセスとインターネットアクセスを区別できます。

4D は IPv4 アドレスを、96-bit の接頭辞付きのハイブリッド型 IPv6/IPv4 フォーマットで返します。たとえば、`::ffff:192.168.2.34` は、`192.168.2.34` という IPv4 アドレスを意味します。詳細については、[IPv6 のサポートについて](#) の章を参照ください。

\$4 - サーバー IP アドレス

\$4 引数は Web サーバーを呼び出すために使用された IP アドレスを受け取ります。4D はマルチホーミングをサポートしており、複数の IP アドレスを持つマシンを使用できます。詳細は [設定ページ](#) を参照ください。

\$5 と \$6 - ユーザー名とパスワード

\$5 と \$6 引数は、ブラウザーが表示する標準の認証ダイアログにユーザーが入力したユーザー名とパスワードを受け取ります。`BASIC` または `DIGEST` 認証が選択されていると、接続のたびにこのダイアログが表示されます。

ブラウザーから送信されたユーザー名が 4D に存在する場合、\$6 引数 (ユーザー名とパスワード) はセキュリティのため渡されません。

\$0 引数

`On Web Authentication` データベースメソッドは布尔値を \$0 に返します：

- \$0=True: 接続を受け入れます。
- \$0=False: 接続を受け入れません。

`On Web Connection` データベースメソッドは、`On Web Authentication` データベースメソッドにより接続が受け入れられた時にのみ実行されます。

警告

\$0 に値が設定されないか、`On Web Authentication` データベースメソッド内で \$0 が定義されていない場合、接続は受け入れられたも

のとされ、On Web Connection データベースメソッドが実行されます。

- On Web Authentication データベースメソッド内でインターフェース要素を呼び出してはいけません（ALERT, DIALOG 等）。メソッドの実行が中断され、接続が拒否されてしまいます。処理中にエラーが発生した場合も同様です。

例題

DIGEST認証モード での On Web Authentication データベースメソッドの例題:

```
// On Web Authentication データベースメソッド
#DECLARE ($url : Text; $header : Text; $ipB : Text; $ipS : Text; \
$user : Text; $pw : Text) -> $valid : Boolean

var $found : cs.WebUserSelection
$valid:=False

$found:=ds.WebUser.query("User === :1";$user)
If($found.length=1) // ユーザーが見つかった場合
    $valid:=WEB Validate digest($user; [WebUser]password)
Else
    $valid:=False // ユーザーは存在しません
End if
```

ユーザーセッション

4D Webサーバーは、ユーザーセッションを管理するビルトインの機能を提供します。ユーザーセッションを作成・維持することで、Webアプリケーション上のユーザー体験を管理・向上することができます。ユーザーセッションが有効かされていると、Webクライアントはリクエスト間で同じコンテキスト（セレクションや変数の値）を再利用できます。

Webサーバーのユーザーセッションでは、以下のことが可能です：

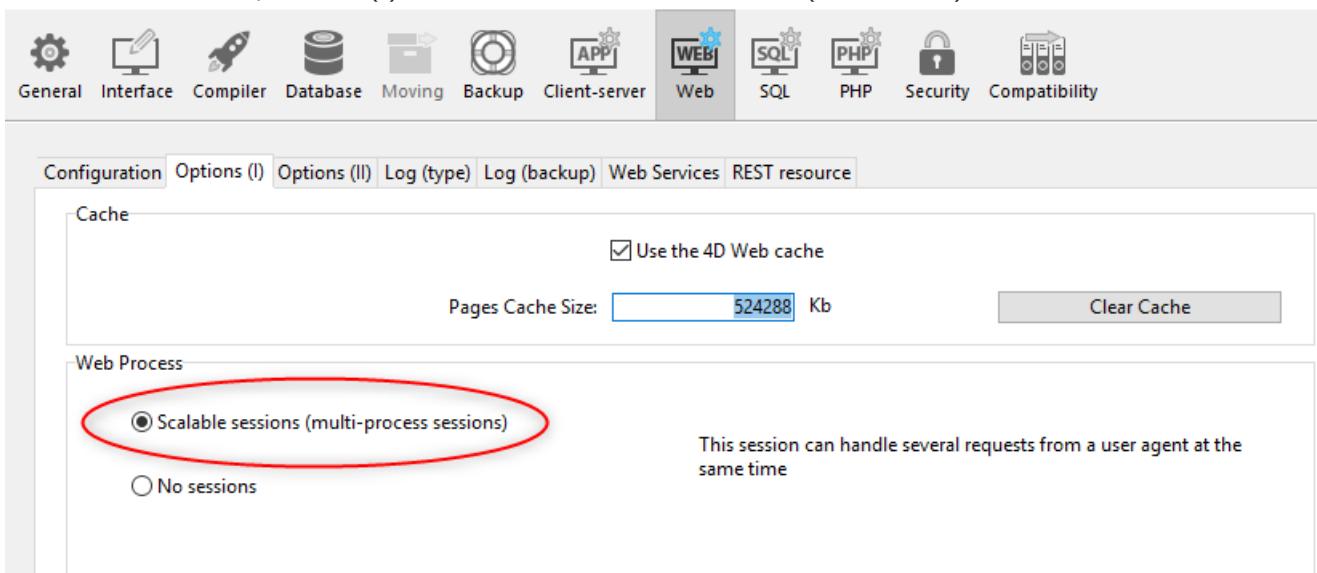
- 同一のWebクライアントからの複数のリクエストを、無制限の並行プロセスで同時に処理（Webサーバーセッションはスケーラブルです）。
- Webクライアントのプロセス間でデータを共有。
- ユーザーセッションに権限を関連付ける。
- `Session` オブジェクトと [Session API](#) を介したアクセスの処理。

注：現在の実装は、Webアプリケーション全体においてセッションを介して階層的なユーザー権限を開発者が管理できるようにする、今後予定されている包括的な機能の最初のステップに過ぎません。

セッションの有効化

セッション管理機能は、4D Webサーバー上で有効または無効にすることができます。セッション管理を有効化する方法は複数あります：

- ストラクチャー設定の `Web / オプション (I)` ページの `スケーラブルセッション` を使用する（永続的な設定）：



このオプションは、新規プロジェクトではデフォルトで選択されています。これは、セッションなし オプションを選択して無効にすることもできます。この場合、Webセッション機能は無効になります（`Session` オブジェクトは使用できません）。

- Webサーバーオブジェクトの `.scalableSession` プロパティを使用する（`.start()` 関数に `settings` 引数として渡します）。この場合、ストラクチャー設定ダイアログボックスで定義されたオプションよりも、Webサーバーオブジェクトの設定が優先されます（ディスクには保存されません）。

メインの Webサーバーのセッションモードは、`WEB SET OPTION` コマンドを使って設定することもできます。

いずれの場合も、設定はマシンに対しローカルなものです。つまり、4D Server の Webサーバーと、リモートの 4Dマシンの Webサーバーで異なる設定が可能です。

互換性について：4D v18 R6 以前の 4Dバージョンで作成されたプロジェクトでは、旧式セッション オプションが使用できます（詳細については、[doc.4d.com](#) の Webサイトを参照ください）。

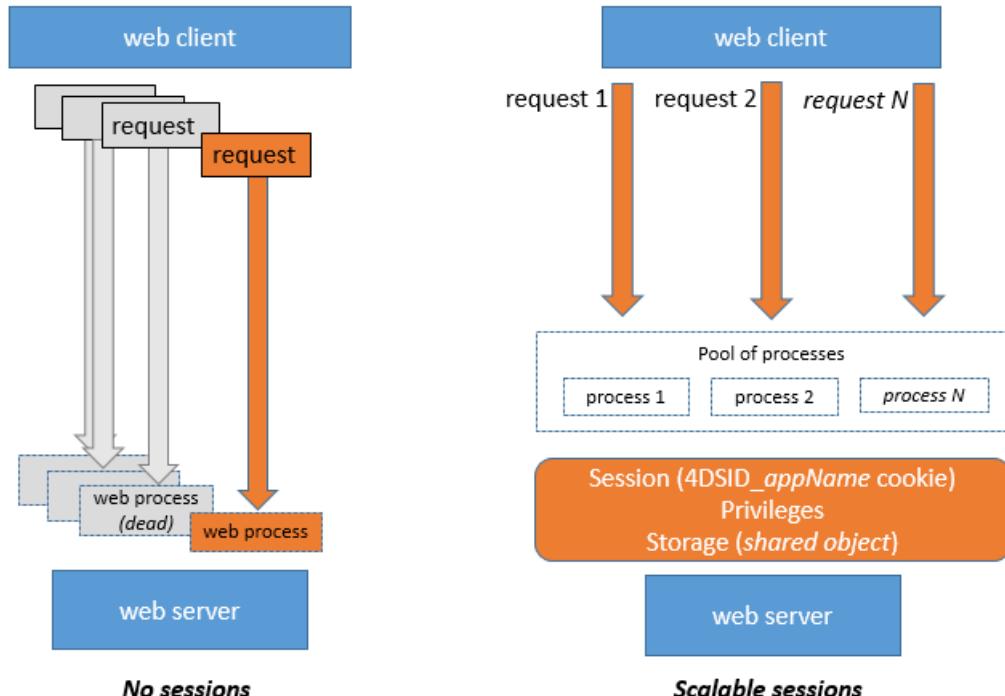
セッションの実装

セッションを有効にする と、4D自身が設定したプライベート cookie ("4DSID_AppName"、AppName はアプリケーションプロジェクトの名称) に基づいて、自動メカニズムが実装されます。この cookie は、アプリケーションのカレントWebセッションを参照します。

この cookie の名前は、`.sessionCookieName` プロパティを使用して取得できます。

1. Webサーバーは、各Webクライアントリクエストにおいて、プライベートな "4DSID_AppName" cookie の存在と値をチェックします。
2. cookie に値がある場合、4D は既存セッションの中からこのクッキーを作成したセッションを探し、見つかった場合には再利用します。
3. クライアントからのリクエストが、すでに開かれているセッションに対応していない場合：
 - プライベートな "4DSID_AppName" cookie を持つ新しいセッションが Webサーバー上に作成されます。
 - 新しいゲスト Session オブジェクトが作成され、このスケーラブルWebセッション専用に使用されます。

カレントの `Session` オブジェクトは、あらゆる Webプロセスのコードにおいて `Session` コマンドを介してアクセスできます。



Webプロセスは通常終了せず、効率化のためにプールされリサイクルされます。プロセスがリクエストの実行を終えると、プールに戻され、次のリクエストに対応できるようになります。Webプロセスはどのセッションでも再利用できるため、実行終了時には（`CLEAR VARIABLE` などを使用し）コードによって `プロセス変数` をクリアする必要があります。このクリア処理は、開かれたファイルへの参照など、プロセスに関連するすべての情報に対して必要です。これが、セッション関連の情報を保持したい場合には、`Session` オブジェクトを使用することが 推奨される理由です。

プリエンプティブモード

4D Server上では、インタプリタモードであっても、Webサーバーセッションは自動的にプリエンプティブプロセスで処理されます。そのため、Webサーバーのコードは `プリエンプティブ実行に準拠` している必要があります。

サーバーマシン上のインターパリターWebコードをデバッグするには、あらかじめサーバーのデバッガーを `サーバー` または `リモートマシン` で有効化する必要があります。これにより、Webプロセスがコオペラティブモードに切り替わり、Webサーバーコードのデバッグが可能になります。

シングルユーザーの 4D では、インターパリターコードは常にコオペラティブモードで実行されます。

情報の共有

各 `Session` オブジェクトには、共有オブジェクトである `.storage` プロパティが用意されています。このプロパティにより、セッションで処理されるすべてのプロセス間で情報を共有することができます。

セッションの有効期限

スケーラブルWebセッションは、以下の場合に閉じられます:

- Webサーバーが停止したとき。
- セッションcookie がタイムアウトしたとき。

非アクティブな cookie の有効期限は、デフォルトでは 60分です。つまり、Webサーバーは、非アクティブなセッションを 60分後に自動的に閉じます。

このタイムアウトは、`Session` オブジェクトの `.idleTimeout` プロパティで設定できます (タイムアウトは 60分未満にはできません)。

スケーラブルWebセッションが閉じられた後に `Session` コマンドが呼び出されると:

- `Session` オブジェクトには権限が含まれていません (ゲストセッション)。
- `.storage` プロパティは空です。
- 新しいセッションcookie がセッションに関連付けられています。

権限

セッションには、権限を関連付けることができます。セッションの権限に応じて、特定のアクセスや機能を Webサーバー上で提供することができます。

権限を割り当てるには、`.setPrivileges()` 関数を使用します。コード内では、`.hasPrivilege()` 関数を使ってセッションの権限をチェックし、アクセスを許可または拒否することができます。デフォルトでは、新しいセッションは権限を持たず、ゲストセッションとなります (`.isGuest()` 関数は `true` を返します)。

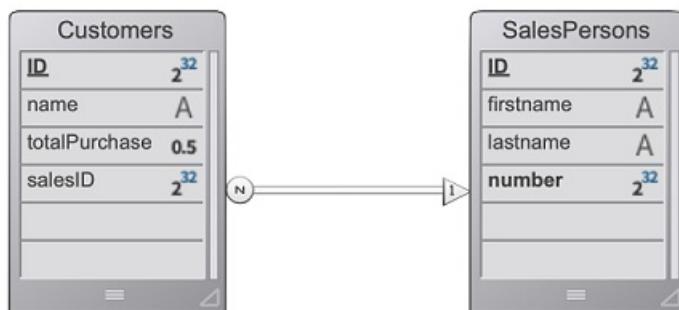
現在の実装では (v18 R6)、"WebAdmin" アクセス権のみ利用可能です。

例:

```
If (Session.hasPrivilege("WebAdmin"))
    // アクセス権が付与されているので、何もしません
Else
    // 認証ページを表示します
End if
```

例題

CRMアプリケーションを使って、各営業担当者が自分の顧客ポートフォリオを管理します。データストアには、少なくとも 2つのリンクされたデータクラス `Customers` と `SalesPersons` が含まれています (営業担当者は複数の顧客を持ちます)。



営業担当者がログインし、Webサーバー上でセッションを開き、上位3名の顧客をセッションに読み込ませたいとします。

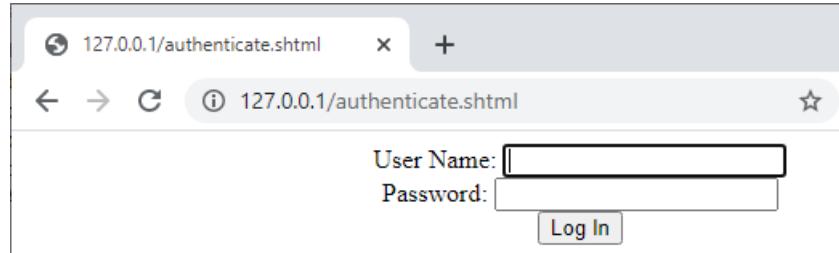
1. セッションを開くために以下の URL を実行します:

`http://localhost:8044/authenticate.shtml`

本番環境では、暗号化されていない情報がネットワーク上を流れるのを防ぐために、[HTTPS接続](#) を使用する必要があります。

2. `authenticate.shtml` ページは、`userId` と `password` の入力フィールドを含むフォームで、4DACTION の POSTアクションを送信します：

```
<!DOCTYPE html>
<html>
<body bgcolor="#ffffff">
<FORM ACTION="/4DACTION/authenticate" METHOD=POST>
    UserId: <INPUT TYPE=TEXT NAME=userId VALUE=""><BR>
    Password: <INPUT TYPE=TEXT NAME=password VALUE=""><BR>
<INPUT TYPE=SUBMIT NAME=OK VALUE="Log In">
</FORM>
</body>
</html>
```



3. authenticate project メソッドは、`userID` に合致する担当者を探し、`SalesPersons` テーブルに保存されているハッシュ値をパスワードと照合します。

```
var $indexUserId; $indexPassword; $userId : Integer
var $password : Text
var $userTop3; $sales; $info : Object

ARRAY TEXT($anames; 0)
ARRAY TEXT($avalue; 0)

WEB GET VARIABLES($anames; $avalue)

$indexUserId:=Find in array($anames; "userId")
$userId:=Num($avalue{$indexUserId})

$indexPassword:=Find in array($anames; "password")
$password:=$avalue{$indexPassword}

$sales:=ds.SalesPersons.query("userId = :1"; $userId).first()

If ($sales#Null)
    If (Verify password hash($password; $sales.password))
        $info:=New object()
        $info.userName:=$sales.firstname+$sales.lastname
        Session.setPrivileges($info)
        Use (Session.storage)
            If (Session.storage.myTop3=NULL)
                $userTop3:=$sales.customers.orderBy("totalPurchase desc").slice(0; 3)
                Session.storage.myTop3:=$userTop3
            End if
        End use
        WEB SEND HTTP REDIRECT("/authenticationOK.shtml")
    Else
        WEB SEND TEXT("This password is wrong")
    End if
Else
    WEB SEND TEXT("This userId is unknown")
End if
```


プリエンプティブWebプロセスの使用

4D Webサーバーを使って、アプリケーションでプリエンプティブWebプロセスを使用することによって、マルチコアコンピューターの利点を最大限引き出すことができます。4D変換タグや Webデータベースメソッド、ORDA の RESTクラス関数を含めた Web関連コードを、可能な限り多くのコアで同時に実行するよう設定することができます。

4D のプリエンプティブプロセスについての詳細は、ランゲージリファレンスの [プリエンプティブ4Dプロセス](#) の章を参照ください。

Webプロセスにおけるプリエンプティブモードの使用可能状況

実行コンテキストによって、プリエンプティブモードが使用される、または使用可能かを次の表に示します：

4D Server	インターパリター (デバッガー有効)	インターパリター (デバッガー無効)	コンパイル済みコード
REST サーバー	コオペラティブ	プリエンプティブ	プリエンプティブ
Web サーバー	コオペラティブ	Web設定	Web設定
Webサービスサーバー	コオペラティブ	Web設定	Web設定

4Dリモート/シングルユーザー	インターパリターコード	コンパイル済みコード
REST サーバー	コオペラティブ	プリエンプティブ
Web サーバー	コオペラティブ	Web設定
Webサービスサーバー	コオペラティブ	Web設定

- REST サーバー: REST で呼び出された [ORDA データモデルクラス関数](#) を処理します
- Web サーバー: [Web テンプレート](#)、[4DACTION とデータベースメソッド](#) を処理します
- Web サービスサーバー: SOAPリクエストを処理します
- Web設定 とは、プリエンプティブモード実行が設定によることを表します：
 - [スケーラブルセッション](#) が選択されている場合、Webプロセスにおいて [プリエンプティブモードが自動的に使用されます](#)。
 - それ以外の場合は、[プリエンプティブプロセスを使用](#) オプションが考慮されます。
 - Webサービスプロセス (サーバーまたはクライアント) のプリエンプティブモードは、メソッドレベルでサポートされています。公開済みの SOAPサーバーメソッド ([4Dで Web サービスを公開する](#) 参照) あるいはプロキシクライアントメソッド ([4Dから Web サービスへサブスクライブする](#) 参照) の "プリエンプティブプロセスで実行可能" プロパティをチェックし、メソッドがコンパイラーによってスレッドセーフと確認されるようにします。

スレッドセーフなWebサーバーコードの書き方

Webプロセスをプリエンプティモードで実行するには、Webサーバーで実行されるすべての 4Dコードがスレッドセーフでなければなりません。[プリエンプティブモードが有効化](#) されている場合、アプリケーションの以下の部分が 4Dコンパイラーによって自動的に評価されます：

- すべての Web関連データベースメソッド：
 - [On Web Authentication](#)
 - [On Web Connection](#)
 - [On REST Authentication](#)
 - [On Mobile App Authentication](#) と [On Mobile App Action](#)
- `compiler_web` プロジェクトメソッド (実際の "実行モード" プロパティに関わらず評価されます)
- Webコンテキストにおいて `PROCESS 4D TAGS` コマンドによって処理される基本的にすべてのコード (.shtmlページを通して実行されるものなど)
- "公開オプション: 4DタグとURL (`4DACTION`)..." 属性が有効なプロジェクトメソッド。
- "RESTリソースとして公開" 属性が有効なテーブルのトリガー
- REST で呼び出された [ORDA データモデルクラス関数](#)

これらそれぞれのメソッドとコードの部分について、スレッドセーフのルールが遵守されているかをコンパイラーがチェックし、問題があった場合にはエラーを返します。スレッドセーフルールについての詳細は、[4Dランゲージリファレンス](#) マニュアルの プロセス の章の [スレッドセーフなメソッドの書き方](#) の段落を参照ください。

4D Webコードのスレッドセーフティ

Web関連のほとんどの 4Dコマンドや関数、データベースメソッド、そして URL がスレッドセーフとなり、プリエンプティモードで使用できます。

4Dコマンドとデータベースメソッド

すべての Web関連コマンドはスレッドセーフです:

- *Web*サーバー テーマの全コマンド
- *HTTP*クライアント テーマの全コマンド

Web関連のデータベースメソッドもスレッドセーフであり、プリエンプティモードで使用することが可能です (前述参照): `On Web Authentication`, `On Web Connection`, `On REST Authentication` ...)。

もちろん、これらのメソッドによって実行されるコードもまたスレッドセーフである必要があります。

WebサーバーURL

以下の 4D WebサーバーURLはスレッドセーフであり、プリエンプティモードで使用可能です:

- `4daction/` (呼び出されるプロジェクトメソッドもまたスレッドセーフでなければいけません)
- `4dcgi/` (呼び出されるデータベースメソッドもまたスレッドセーフでなければいけません)
- `4dwebtest/`
- `4dblank/`
- `4dstats/`
- `4dhtmlstats/`
- `4dcache-clear/`
- `rest/`
- `4dimgfield/` (ピクチャーフィールドの Webリクエストに対し `PROCESS 4D TAGS` によって生成されます)
- `4dimg/` (ピクチャー変数の Webリクエストに対し `PROCESS 4D TAGS` によって生成されます)

プリエンプティブWebプロセスアイコン

ランタイムエクスプローラーと 4D Server管理ウィンドウの両方において、プリエンプティブな Webプロセスに対し専用アイコンが表示されるようになりました:

プロセスタイプ	アイコン
プリエンプティブWebメソッド	

はじめに

4D は、4D アプリケーションに格納されているデータへのダイレクトアクセスを可能にする強力な REST サーバーを提供しています。

REST サーバーは 4D および 4D Server に含まれており、[設定完了後は 4D アプリケーションにて自動的に利用可能となります。](#)

この章では、簡単な例題を使用して REST 機能を紹介します。これから、実際に次のことをしてみましょう：

- 簡単な 4D アプリケーションプロジェクトを作成し、設定します。
- 標準のブラウザーを開き、REST を介して 4D プロジェクトのデータにアクセスします。

例題が複雑にならないよう、ここでは 4D とブラウザーを同じマシン上で使用します。もちろん、リモートアーキテクチャーを使うことも可能です。

4D プロジェクトの作成と設定

1. 4D または 4D Server アプリケーションを起動し、新規プロジェクトを作成します。名前は仮に "Emp4D" とします。

2. ストラクチャーエディターを開き、[Employees] テーブルを作成して、次のフィールドを追加します：

- Lastname (文字列)
- Firstname (文字列)
- Salary (倍長整数)

Employees	
ID	2 ³²
Lastname	A
Firstname	A
Salary	2 ³²

テーブルおよび各フィールドの "RESTリソースとして公開" オプションはデフォルトで選択されています。これを変更しないでください。

3. フォームを作成し、何名かの社員レコードを作成します：

ID :	Lastname :	Firstname :	Salary :
1	Brown	Michael	25000
2	Jones	Maryanne	35000
3	Smithers	Jack	41000

4. ストラクチャーセットの Web > Web機能 ページを開き、[REST サーバーとして公開](#) オプションを選択します。

5. 上部の 実行 メニューから、必要に応じて Web サーバー開始 を選択し、次に同メニューから Web サーバーテスト を選択します。

規定のブラウザーが開かれ、4D Web サーバーのデフォルトホームページが表示されます。

ブラウザーから 4D データにアクセスする

これで、RESTリクエストを使った 4D のデータの読み込み・編集が可能になりました。

4D の REST URL リクエストは必ず、address:port エリアの後に入る /rest から始まります。たとえば、4D データストアの内容を確認するには、次のように書けます：

```
http://127.0.0.1/rest/$catalog
```

RESTサーバーの応答です:

```
{  
    "_UNIQID": "96A49F7EF2ABDE44BF32059D9ABC65C1",  
    "dataClasses": [  
        {  
            "name": "Employees",  
            "uri": "/rest/$catalog/Employees",  
            "dataURI": "/rest/Employees"  
        }  
    ]  
}
```

これは、データストアに Employees データクラスが格納されていることを意味します。データクラス属性を確認するには、次のように書きます:

```
/rest/$catalog/Employees
```

また、Employees データクラスの全エンティティを取得するには:

```
/rest/Employees
```

レスポンス:

```
{
  "__entityModel": "Employees",
  "__GlobalStamp": 0,
  "__COUNT": 3,
  "__FIRST": 0,
  "__ENTITIES": [
    {
      "__KEY": "1",
      "__TIMESTAMP": "2020-01-07T17:07:52.467Z",
      "__STAMP": 2,
      "ID": 1,
      "Lastname": "Brown",
      "Firstname": "Michael",
      "Salary": 25000
    },
    {
      "__KEY": "2",
      "__TIMESTAMP": "2020-01-07T17:08:14.387Z",
      "__STAMP": 2,
      "ID": 2,
      "Lastname": "Jones",
      "Firstname": "Maryanne",
      "Salary": 35000
    },
    {
      "__KEY": "3",
      "__TIMESTAMP": "2020-01-07T17:08:34.844Z",
      "__STAMP": 2,
      "ID": 3,
      "Lastname": "Smithers",
      "Firstname": "Jack",
      "Salary": 41000
    }
  ],
  "__SENT": 3
}
```

取得するデータを様々な条件でフィルターすることも可能です。たとえば、2番目のエンティティの "Lastname" 属性値のみを取得するには、次のように書きます:

```
/rest/Employees(2)/Lastname
```

レスポンス:

```
{
  "__entityModel": "Employees",
  "__KEY": "2",
  "__TIMESTAMP": "2020-01-07T17:08:14.387Z",
  "__STAMP": 2,
  "Lastname": "Jones"
}
```

4D の REST API は、4Dアプリケーションを操作するためのコマンドを多数提供しています。

サーバー設定

4D の RESTサーバーは、標準の HTTPリクエストを用いて外部アプリケーションがアプリケーションのデータにアクセスすることを可能にします。つまり、プロジェクトのデータクラス情報を取得したり、データを操作したり、Webアプリケーションにログインしたり、といったことが可能です。

REST機能を使い始めるまえに、まずは 4D REST サーバーの設定をおこない、これを起動させる必要があります。

- 4D Server上では、開かれる RESTセッションにつき、4D Client ライセンスが1消費されます。
- シングルユーザーの4D上では、テスト目的で RESTセッションを 3つ開くことができます。
- リクエストをおこなうアプリケーションの [セッション](#) は別途管理する必要があります。

RESTサーバーを開始する

セキュリティ上の理由により、デフォルトでは、4D は RESTリクエストに応答しません。RESTサーバーを開始し、RESTリクエストを処理するには、ストラクチャー設定 の Web > Web機能 ページにて、RESTサーバーとして公開 オプションを有効化する必要があります。

The screenshot shows the 'Git - Structure Settings' interface. The top navigation bar includes icons for General, Interface, Compiler, Database, Moving, Backup, Client-server, Web (selected), SQL, PHP, Security, and Compatibility. Below this is a tab bar with Configuration, Options (I), Options (II), Log (type), Log (backup), Web Services, and Web features (selected). The main area is titled 'Publishing' and contains a note: 'NOTE: This service is only active on 4D Developer Professional and 4D Server.' A checkbox labeled 'Expose as REST server' is checked. The 'Access' section contains a note: 'NOTE: This setting is only taken into account when the 4D password access system is activated (the Designer has been assigned a password) and the database method "On REST Authentication" does not exist.' A dropdown menu for 'Read/Write:' shows '<Anyone>'.

RESTサービスは 4D の HTTPサーバーを使用するため、4D Webサーバーが開始されていることを確認してください。

このオプションが有効化されると、「警告: アクセス権が正しく設定されているか確認してください。」という警告メッセージが表示されます。これは REST接続の認証設定がされていない限り、デフォルトではデータベースオブジェクトに自由にアクセスできてしまうためです。

変更を反映するには、4Dアプリケーションを再起動する必要があります。

アクセス権の設定

デフォルトでは、REST接続はすべてのユーザーに対してオープンですが、この状態はライセンス管理上もセキュリティ上も推奨されません。

REST接続は次の方法で制限することができます:

- ストラクチャー設定の "Web > Web機能" ページにて、RESTサービスに割り当てる 読み込み/書き出し ユーザーグループを設定します;
- `On REST Authentication` データベースメソッドに、RESTの初期リクエストを処理するコードを書きます。

上に挙げた 2つの方法を同時に使用することはできません。`On REST Authentication` データベースメソッドを定義した場合、4D は RESTリクエストの処理を同メソッドに委ねます。つまり、ストラクチャー設定の "Web > Web機能" ページにて指定した "読み込み/書き出し" の設定は無視されます。

ストラクチャー設定を使用する

ストラクチャー設定の "Web > Web機能" ページにある 読み込み/書き出し 設定は、RESTクリエイティブを使って 4Dアプリケーションへのリンクを設立するとのできる 4Dユーザーのグループを指定します。

デフォルトでは、メニューには <Anyone> が選択されています。これは、REST接続はすべてのユーザーに対してオープンであるという状態を示しています。グループを指定すると、そのグループに所属する 4Dユーザー アカウントのみが RESTリクエストを通して 4D にアクセス できるようになります。このグループに所属していないアカウントの場合、4D はリクエストの送信者に対して認証エラーを返します。

この設定を使用するには、On REST Authentication データベースメソッドを定義してはいけません。これが定義されている場合は、ストラクチャー設定にて指定したアクセス設定は無視されます。

On REST Authentication データベースメソッドを使用する

On REST Authentication データベースメソッドは 4D 上で RESTセッションの開始を管理するための方法を提供します。RESTリクエストによって新規セッションが開始される際、このデータベースメソッドは自動的に呼び出されます。RESTセッション開始のリクエスト を受信すると、そのリクエストヘッダーには接続の識別子が含まれています。これらの識別子を評価するために On REST Authentication データベースメソッドは呼び出されます。評価にあたっては、4Dアプリケーションのユーザーリストを使用することもできますし、独自の識別子のテーブルを使用することもできます。詳細については On REST Authentication データベースメソッドの [ドキュメンテーション](#) を参照ください。

テーブルやフィールドの公開

4Dアプリケーションの RESTサービスが有効化されると、[データストアインターフェース](#) を通して 4Dデータベースのすべてのテーブルとフィールドおよび格納データが RESTセッションによってデフォルトでアクセス可能です。つまり、すべてのデータにアクセス可能ということです。たとえば、データベースに [Employee] テーブルが含まれている場合、次のように書くことができます：

```
http://127.0.0.1:8044/rest/Employee/?$filter="salary>10000"
```

このリクエストで、salary (給与) フィールドが 10000以上 の社員データが取得されます。

"非表示" 属性を選択されたテーブルやフィールドも、デフォルトで REST に公開されています。

REST 経由でアクセス可能なデータストアオブジェクトを制限するには、アクセス不可にするテーブルやフィールドについて "RESTリソースとして公開" オプションを選択解除する必要があります。許可されていないリソースへの RESTリクエストがあった場合、4Dはエラーを返します。

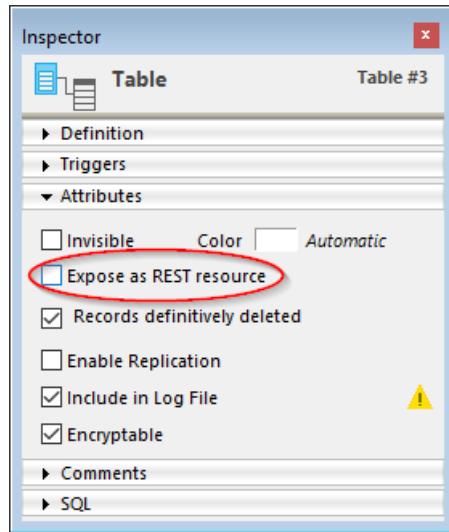
テーブルの公開

デフォルトでは、すべてのテーブルが REST に公開されています。

セキュリティ上の理由から、データベースの一部のテーブルのみを公開したい状況もあるでしょう。たとえば、[Users] テーブルを作成し、その中にユーザー名とパスワードが保存されている場合、そのテーブルは公開しない方が賢明でしょう。

テーブルを公開したくない場合は：

1. ストラクチャーエディターにて対象となるテーブルを選択し、右クリックでコンテキストメニューを開いてテーブルプロパティを選択します。
2. RESTリソースとして公開 オプションの選択を解除します：



公開設定を変更する各テーブルに対して、この手順を繰り返します。

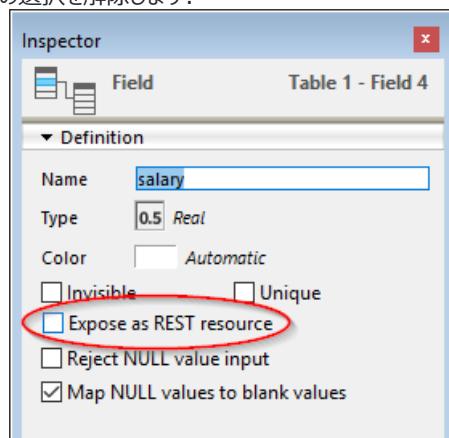
フィールドの公開

デフォルトでは、すべての 4Dデータベースフィールドが REST に公開されています。

テーブルの一部のフィールドのみを非公開にしたい状況もあるでしょう。たとえば、[Employees]Salary のようなフィールドは非公開の方がよいでしょう。

フィールドを非公開にするには：

1. ストラクチャーエディターにて対象となるフィールドを選択し、右クリックでコンテキストメニューを開いてフィールドプロパティを選択します。
2. フィールドの RESTリソースとして公開 オプションの選択を解除します：



公開設定を変更する各フィールドに対して、この手順を繰り返します。

あるフィールドが REST を通してアクセス可能であるためには、その親テーブルも公開されている必要があります。親テーブルが公開されていない場合、各フィールドの公開設定に関わらず、すべてのフィールドがアクセス不可になります。

ユーザーとセッション

RESTリクエストは [Webユーザーセッション](#) の恩恵を受けることができます。これにより、複数リクエストの処理や、Webクライアントプロセス間のデータ共有、ユーザー権限などの追加機能を利用することができます。

4D Server上で RESTセッションを開くには、まずリクエストを送信するユーザーが認証されなければなりません。

ユーザー認証

アプリケーションにユーザーをログインするには、ユーザー名とパスワードをヘッダーに含めた POSTリクエスト内で `$directory/login` を呼び出します。このリクエストは `On REST Authentication` データベースメソッド (存在すれば) を呼び出します。このメソッド内でユーザーの認証をおこなうことができます (後述参照)。

セッションの開始

[スケーラブルセッションを有効化](#) (推奨) している場合に、`On REST Authentication` データベースメソッドが `true` を返すと、ユーザー SESSIONは自動的に開かれ、`Session` オブジェクトおよび [Session API](#) を介して管理することができます。後続の RESTリクエストは同じセッションcookieを使用します。

`On REST Authentication` データベースメソッドが定義されてない場合には、`guest` セッションが開かれます。

例題

この例では、ユーザーが htmlページにメールアドレスとパスワードを入力し、POST で `$directory/login` をリクエストします (htmlページの送信においては、HTTPS接続の使用が推奨されます)。これによって呼び出された `On REST Authentication` データベースメソッドがユーザー認証をおこない、セッションを確立します。

htmlログインページ:

The form consists of three elements: a label 'Email:' followed by an input field, a label 'Password:' followed by an input field, and a blue 'Login' button.

```

<html><body bgcolor="#ffffff">

<div id="demo">
  <FORM name="myForm">
    メールアドレス: <INPUT TYPE=TEXT NAME=userId VALUE=""><BR>
    パスワード: <INPUT TYPE=TEXT NAME=password VALUE=""><BR>
    <button type="button" onclick="onClick()">
      ログイン
    </button>
  </div id="authenticationFailed" style="visibility:hidden;">ログインに失敗しました</div>
</FORM>
</div>

<script>
function sendData(data) {
  var XHR = new XMLHttpRequest();

  XHR.onreadystatechange = function() {
    if (this.status == 200) {
      window.location = "authenticationOK.shtml";
    }
    else {
      document.getElementById("authenticationFailed").style.visibility = "visible";
    }
  };
}

XHR.open('POST', 'http://127.0.0.1:8044/rest/$directory/login'); // RESTサーバーアドレス

XHR.setRequestHeader('username-4D', data.userId);
XHR.setRequestHeader('password-4D', data.password);
XHR.setRequestHeader('session-4D-length', data.timeout);

XHR.send();
};

function onClick()
{
sendData({userId:document.forms['myForm'].elements['userId'].value , password:document.forms['myForm'].e
}
</script></body></html>

```

サーバーにログイン情報が送信されると、`On REST Authentication` データベースメソッドが呼び出されます:

```

// On REST Authentication データベースメソッド

#DECLARE($userId : Text; $password : Text) -> $Accepted : Boolean
var $sales : cs.SalesPersonsEntity

$Accepted:=False

// ヘッダーに username-4D と password-4D を含めて '/rest' URL が呼び出されました
If ($userId#"")
  $sales:=ds.SalesPersons.query("email = :1"; $userId).first()
  If ($sales#Null)
    If (Verify password hash($password; $sales.password))
      fillSession($sales)
      $Accepted:=True
    End if
  End if
End if

```

一旦呼び出されて `True` を返すと、同セッションにおいて `On REST Authentication` データベースメソッドはそれ以上呼び出されません。

`fillSession` プロジェクトメソッドは、たとえば次のようにユーザー SESSION を初期化します：

```
#DECLARE($sales : cs.SalesPersonsEntity)
var $info : Object

$info:=New object()
$info.userName:=$sales.firstname" "+$sales.lastname

Session.setPrivileges($info)

Use (Session.storage)
If (Session.storage.myTop3=NULL)
    Session.storage.myTop3:=$sales.customers.orderBy("totalPurchase desc").slice(0; 3)
End if
End use
```

サーバー情報の取得

RESTサーバーの次の情報を取得することができます:

- 公開されているデータクラスとデータクラス属性
- RESTサーバーのキャッシュの中身 (ユーザーセッションを含む)

カタログ

公開されているデータクラスとデータクラス属性 のリストを取得するには `$catalog`、`$catalog/{dataClass}`、または `$catalog/$all` パラメーターを使います。

公開されている全データクラスとデータクラス属性のコレクションを取得するには:

```
GET /rest/$catalog/$all
```

キャッシュ情報

4D Server のキャッシュに保存されているエンティティセレクション、および実行中のユーザーセッションの情報を取得するには `$info` パラメーターを使います。

queryPath と queryPlan

クエリによって生成されたエンティティセレクションは、`queryPlan` と `queryPath` という 2つのプロパティを持ちます。これらのプロパティを算出・取得するには、RESTリクエストに `$queryPlan` および `$queryPath` を追加します。

たとえば:

```
GET /rest/People/$filter="employer.name=acme AND lastName=Jones"&$queryplan=true&$querypath=true
```

これらのプロパティは、データクラスやリレーションに対する複合クエリをサーバーが内部的にどのようにおこなっているかの情報を格納するオブジェクトです:

- `queryPlan`: 実行前のクエリについての詳細な情報 (クエリプラン) を格納するオブジェクト。
- `queryPath`: 実際に実行されたクエリ処理の詳細な情報 (クエリパス) を格納するオブジェクト。

情報には、クエリの種類 (インデックスあるいはシーケンシャル)、必要なサブクエリおよびその連結演算子が含まれます。クエリパスには、見つかったエンティティの数と各検索条件を実行するにかかる時間も含まれます。この情報は、アプリケーションの開発中に解析することで有効に活用できます。一般的には、クエリプランとクエリパスの詳細は同一になるはずですが、4D はパフォーマンスの向上のために、動的な最適化をクエリ実行時に実装することがあるからです。たとえば、その方が早いと判断した場合には、4Dエンジンはインデックス付きクエリをシーケンシャルなものへと動的に変換することができます。これは検索されているエンティティの数が少ないとときに起こります。

データ操作

REST によって、すべての [公開されているデータクラス、属性](#)、そして [関数](#) にアクセスすることができます。データクラス、属性、および関数名については、文字の大小が区別されます。クエリのデータについては、文字の大小は区別されません。

データのクエリ

データを直接クエリするには `$filter` 関数を使います。たとえば、"Smith" という名前の人を検索するには:

```
http://127.0.0.1:8081/rest/Person/?$filter="lastName=Smith"
```

エンティティの追加・編集・削除

REST API を使って、4D内と同等のデータ操作をおこなうことができます。

エンティティを追加・編集するには `$method=update` を呼び出します。1つ以上のエンティティを削除するには `$method=delete` を使用します。

`{dataClass}({key})` でデータクラスのいちエンティティを取得する以外にも、エンティティセレクションやコレクションを返す [クラス関数](#) を用意することもできます。

戻り値としてセレクションを返す前に、`$orderby` を使って一つ以上の属性（リレーション属性も可）を基準に並べ替えることもできます。

データのナビゲーション

エンティティのコレクションをナビゲートするにあたっては、クエリやエンティティセレクションに次の RESTリクエストを追加することができます: `$skip` (開始エンティティの指定)、`$top/$limit` (返されるエンティティ数の指定)。

エンティティセットの作成と管理

エンティティセットとは、エンティティセレクション と同等の意味で、RESTリクエストによって取得され、4D Server のキャッシュに保存されるエンティティのコレクションのことです。エンティティセットを利用することで、同じ結果を得るためにアプリケーションを繰り返しクエリすることが避けられます。エンティティセットへのアクセスはクエリするよりも速いため、アプリケーション速度の向上にもつながります。

エンティティセットを作成するには、RESTリクエスト内で `$method=entityset` を呼び出します。エンティティセットがタイムアウトした場合やサーバーから削除されてしまった場合への安全対策として、`$filter` や `$orderby` を呼び出す際に `$savedfilter` および `$savedorderby` を使用することで、以前と同じ ID で再取得することができます。

エンティティセットにアクセスするには、`$entityset/{entitySetID}` を使います。例:

```
/rest/People/$entityset/0AF4679A5C394746BFEB68D2162A19FF
```

デフォルトで、エンティティセットは 2時間保存されます。`$timeout` に新しい値を渡すことで、タイムアウトを変更できます。エンティティセットを使用するたびに、タイムアウトはデフォルト値または指定値にリセットされます。

4D Server のキャッシュからエンティティセットを削除したい場合には `$method=release` を使います。

エンティティセット内のエンティティの属性値を編集すると、それらの値が更新されます。ただし、エンティティセットの生成に使用したクエリ条件に合致する値から合致しない値に変更したとしても、そのエンティティはエンティティセットから削除されません。エンティティを削除した場合には、エンティティセットからも削除されます。

4D Server のキャッシュからエンティティセットが消えていた場合、10分のデフォルトタイムアウトで再作成されます。エンティティセットが消えていた場合、再作成されるエンティティセットの内容は更新されたものです（新しくエンティティが追加されていたり、存在していたエンティティが削除されていたりする場合があります）。

`$entityset/{entitySetID}?$logicOperator... &$otherCollection` を使って、事前に作成した 2つのセンティティセットを統合できます。両セットの内容を統合する（集合の和）ほか、共通のエンティティのみを返したり（集合の積）、共通でないエンティティのみを返したり（集合の対称差）することができます。

この場合新規のエンティティセレクションが返されます。RESTリクエストの最後に `$method=entityset` を追加することで新規のエンティティセットを作成することもできます。

データの計算

`$compute` を使って、データクラスの任意の属性について、`average`や `count`、`min`、`max`、`sum` といった計算がおこなえます。`$all` キーワードを使えば、全種の値を計算できます。

たとえば、一番高い給与を取得するには:

```
/rest/Employee/salary/?$compute=max
```

全種の値を計算して JSONオブジェクトとして返すには:

```
/rest/Employee/salary/?$compute=$all
```

データモデルクラス関数の呼び出し

POSTリクエストを使って、ORDAデータモデルの [ユーザークラス関数](#) を呼び出すことで、ターゲットアプリケーションの公開APIを活用できます。たとえば、City DataClassクラスに `getCity()` 関数を定義した場合、次のリクエストで呼び出すことができます:

```
/rest/City/getCity
```

データはリクエストボディに含めます: `["Paris"]`

RESTサービスとして公開された 4Dプロジェクトメソッドへの呼び出しありは引き続きサポートされていますが、廃止予定となっています。

取得する属性の選択

RESTレスポンスにどの属性を含めて返してもらうかを指定するには、初期リクエストに属性のパスを追加します (例: `Company(1)/name, revenues/`)。

このフィルターは次の方法で適用できます:

オブジェクト	シンタックス	例題
データクラス	{dataClass}/{att1,att2...}	/People/firstName,lastName
エンティティのコレクション	{dataClass}/{att1,att2...}/?\$filter="{filter}"	/People/firstName,lastName/?\$filter="lastName='a@'"
特定のエンティティ	{dataClass}({ID})/{att1,att2...}	/People(1)/firstName,lastName
	{dataClass}:{attribute}(value)/{att1,att2...}/	/People:firstName(Larry)/firstName,lastName/
エンティティセレクション	{dataClass}/{att1,att2...}/\$entityset/{entitySetID}	/People/firstName/\$entityset/528BF90F10894915A429

属性名はコンマ区切りで渡します (例: `/Employee/firstName,lastName,salary`)。ストレージ属性およびリレーション属性を渡すことができます。

例題

エンティティを取得する際に、レスポンスに含める属性を指定する例をいくつか紹介します。

この方法は次を対象に使用できます:

- データクラス (データクラスの全エンティティまたはエンティティのコレクション)
- 特定のエンティティ
- エンティティセット

データクラスの例

次のリクエストは、People データクラス (データクラス全体または `$filter` の定義に応じたエンティティセレクション) から名字 (firstName) と名前 (lastName) 属性のみを取得します。

```
GET /rest/People/firstName,lastName/
```

結果:

```
{
  __entityModel: "People",
  __COUNT: 4,
  __SENT: 4,
  __FIRST: 0,
  __ENTITIES: [
    {
      __KEY: "1",
      __STAMP: 1,
      firstName: "John",
      lastName: "Smith"
    },
    {
      __KEY: "2",
      __STAMP: 2,
      firstName: "Susan",
      lastName: "O'Leary"
    },
    {
      __KEY: "3",
      __STAMP: 2,
      firstName: "Pete",
      lastName: "Marley"
    },
    {
      __KEY: "4",
      __STAMP: 1,
      firstName: "Beth",
      lastName: "Adams"
    }
  ]
}
```

GET /rest/People/firstName,lastName/?\$filter="lastName='A@'"/

結果:

```
{
  __entityModel: "People",
  __COUNT: 1,
  __SENT: 1,
  __FIRST: 0,
  __ENTITIES: [
    {
      __KEY: "4",
      __STAMP: 4,
      firstName: "Beth",
      lastName: "Adams"
    }
  ]
}
```

特定エンティティの例

次のリクエストは、People データクラスの特定エンティティについて、名字 (firstName) と名前 (lastName) 属性のみを取得します。

GET /rest/People(3)/firstName,lastName/

結果:

```
{  
    __entityModel: "People",  
    __KEY: "3",  
    __STAMP: 2,  
    firstName: "Pete",  
    lastName: "Marley"  
}
```

```
GET /rest/People(3)/
```

結果:

```
{  
    __entityModel: "People",  
    __KEY: "3",  
    __STAMP: 2,  
    ID: 3,  
    firstName: "Pete",  
    lastName: "Marley",  
    salary: 30000,  
    employer: {  
        __deferred: {  
            uri: "http://127.0.0.1:8081/rest/Company(3)",  
            __KEY: "3"  
        }  
    },  
    fullName: "Pete Marley",  
    employerName: "microsoft"  
}
```

エンティティセットの例

[エンティティセットの作成](#) 後に、どの属性を返すかを指定して、エンティティセットの情報をフィルターできます:

```
GET /rest/People/firstName,employer.name/$entityset/BD_CD8AABE13144118A4CF8641D5883F5?$expand=employer
```

画像属性の表示

画像属性の全体像を表示させるには、次のように書きます:

```
GET /rest/Employee(1)/photo?$imageformat=best&$version=1&$expand=photo
```

画像形式についての詳細は [\\$imageformat](#) を参照ください。version パラメーターについての詳細は [\\$version](#) を参照ください。

BLOB属性のディスク保存

データクラスに保存されている BLOB をディスクに保存するには、次のように書きます:

```
GET /rest/Company(11)/blobAtt?$binary=true&$expand=blobAtt
```

1件のエンティティの取得

エンティティを 1件のみ取得したい場合には [{dataClass}:{attribute}\(value\)](#) シンタクスを利用できます。これは、データクラスの主キーに基づかないリレーション検索をしたい場合に便利です。たとえば:

```
GET /rest/Company:companyCode("Acme001")
```


ORDAクラス関数の呼び出し

RESTリクエストを使って、ORDAデータモデルに定義されている [データモデルクラス関数](#) を呼び出すことで、ターゲット 4Dアプリケーションの公開APIを活用できます。

関数を呼び出すには () を省き、適切な ORDA のコンテキストにおいて POST リクエストでおこないます。たとえば、City DataClassクラスに `getCity()` 関数を定義した場合、次のリクエストで呼び出すことができます:

```
/rest/City/getCity
```

POST リクエストのボディに関数に渡す引数を含めます: `["Aguada"]`

この呼び出しは、4Dランゲージでは次のステートメントに相当します:

```
$city:=ds.City.getCity("Aguada")
```

RESTリクエストで直接呼び出せるのは `exposed` キーワードが付いた関数のみです。 [公開vs非公開関数](#) の章を参照ください。

関数の呼び出し

関数は必ず REST の POST リクエストで呼び出さなくてはなりません (GETリクエストの場合はエラーが返されます)。

サーバーのデータストアの対応するオブジェクトを対象に、関数は呼び出されます。

クラス関数	シンタックス
DataStore クラス	<code>/rest/\$catalog/DataStoreClassFunction</code>
DataClass クラス	<code>/rest/{dataClass}/DataClassClassFunction</code>
EntitySelection クラス	<code>/rest/{dataClass}/EntitySelectionClassFunction</code>
	<code>/rest/{dataClass}/EntitySelectionClassFunction/\$entityset/entitySetNumber</code>
	<code>/rest/{dataClass}/EntitySelectionClassFunction/\$filter</code>
	<code>/rest/{dataClass}/EntitySelectionClassFunction/\$orderby</code>
Entity クラス	<code>/rest/{dataClass}(key)/EntityClassFunction/</code>

`/rest/{dataClass}/Function` は DataClassクラスまたは EntitySelectionクラスの関数を呼び出すのに使えます (`/rest/{dataClass}` はデータクラスの全エンティティをエンティティセレクションに返します)。

EntitySelection クラスの関数が先に探されます。見つからない場合に、DataClassクラスを探します。つまり、同じ名称の関数が DataClassクラスと EntitySelectionクラスの両方に定義されている場合、DataClassクラスの関数が実行されることはありません。

プロジェクトがコンパイル済みモードで実行される場合、RESTサーバーは常にプリエンプティブプロセスを使用するため、RESTリクエストから呼び出されるすべての 4Dコードは スレッドセーフでなければなりません ([プリエンプティブプロセスを使用の設定値](#) は、RESTサーバーによって無視されます)。

引数

ORDAユーザークラスに定義された関数には、引数を渡すことができます。サーバーサイドでこれらの引数は、クラス関数の \$1, \$2 などのパラメーターに受け渡されます。

次のルールが適用されます:

- 引数はすべて、POSTリクエストのボディに渡す必要があります:
- 引数はコレクション (JSON形式) の中に格納する必要があります。
- JSON コレクションがサポートしているスカラーなデータ型はすべて引数として渡せます。
- エンティティやエンティティセレクションも引数として受け渡せます。この際、対応する ORDAオブジェクトにデータを割り当てるために RESTサーバーが使用的する専用の属性 (`__DATACLASS`, `__ENTITY`, `__ENTITIES`, `__DATASET`) を JSONオブジェクトに含めなければなりません。

[エンティティを引数として受け取る例題](#) と [エンティティセレクションを引数として受け取る例題](#) を参照ください。

スカラー値の引数

引数は、ボディに定義されたコレクションに格納します。たとえば、DataClass クラス関数 `getCities()` はがテキスト引数を受け取る場合:

```
/rest/City/getCities
```

ボディの引数: `["Aguada", "Paris"]`

引数としてサポートされるのは、JSONポインターを含むすべての JSON のデータ型です。日付は ISO 8601形式の文字列として渡せます (例: `"2020-08-22T22:00:00Z"`)。

エンティティ引数

引数として渡されたエンティティは、キー (`__KEY` プロパティ) によってサーバー上で参照されます。リクエストにおいてキーが省略されていれば、サーバーのメモリに新規エンティティが読み込まれます。エンティティが持つ属性について、値を受け渡すことも可能です。サーバー上でこれらの値は自動的に当該エンティティ用に使用されます。

サーバー上の既存エンティティについて変更された属性値をリクエストが送信した場合、呼び出した ORDAデータモデル関数は自動的に変更後の値で実行されます。この機能によって、たとえばエンティティに対する処理の、すべてのビジネスルールを適用した後の結果をクライアントアプリケーションから確認することができます。その結果をもとにエンティティをサーバー上で保存するかどうかを判断できます。

プロパティ	タイプ	説明
エンティティの属性	混合	任意 - 変更する値
<code>__DATACLASS</code>	String	必須 - エンティティのデータクラスを指定します
<code>__ENTITY</code>	Boolean	必須 - <code>true</code> は引数がエンティティであることをサーバーに通知します
<code>__KEY</code>	混合 (プライマリーキーと同じ型)	任意 - エンティティのプライマリーキー

- `__KEY` が省略された場合、指定した属性を持つ新規エンティティがサーバー上で作成されます。
- `__KEY` が提供された場合、`__KEY` が合致するエンティティが指定した属性とともにサーバー上に読み込まれます。

エンティティを [作成](#) または [更新](#) する例題を参照ください。

リレートエンティティ引数

[エンティティ引数](#) と同じプロパティを持ちます。異なる点は、リレートエンティティは存在していなければならないため、プライマリーキーを格納する `__KEY` を省略できません。

リレートエンティティを持つエンティティを [作成](#) または [更新](#) する例題を参照ください。

エンティティセレクション引数

引数として渡すエンティティセレクションはあらかじめ `$method=entityset` によって定義されている必要があります。

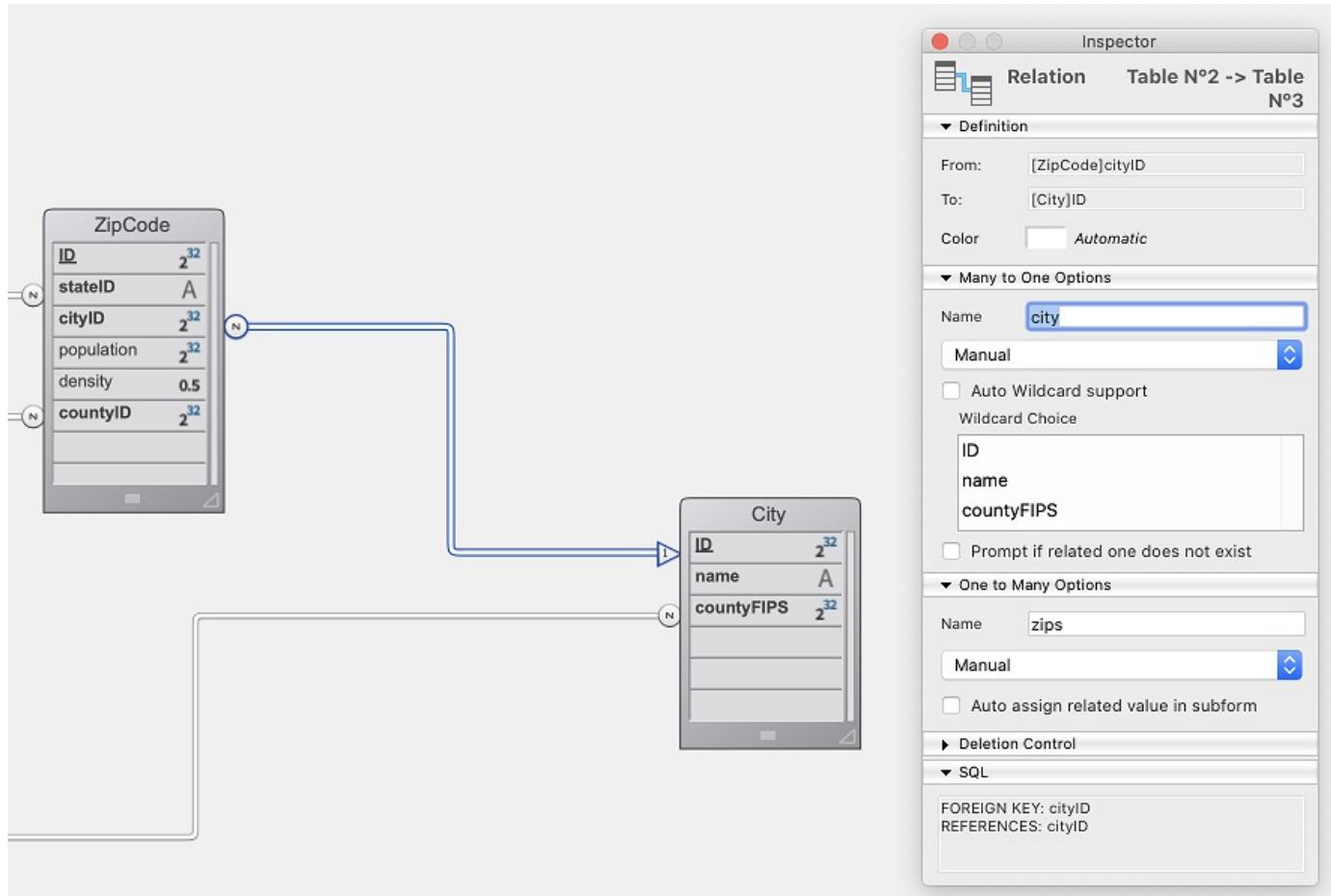
変更されたエンティティセレクションをリクエストがサーバーに送信した場合、呼び出した ORDAデータモデル関数は自動的に変更後のエンティティセレクションで実行されます。

プロパティ	タイプ	説明
エンティティの属性	混合	任意 - 変更する値
__DATASET	String	必須 - エンティティセレクションのエンティティセットID (UUID)
__ENTITIES	Boolean	必須 - true は引数がエンティティセレクションであることをサーバーに通知します

[エンティティセレクションを引数として受け取る例題](#) を参照ください。

リクエストの例題

このデータベースは、localhost (ポート8111) 上でリモートデータストアとして公開されています。



データストアクラス関数を使用する

US_Cities DataStore クラスは API を提供しています:

```
// cs.DataStore クラス

Class extends DataStoreImplementation

exposed Function getName()
$0:="US cities and zip codes manager"
```

次のリクエストを実行します:

POST 127.0.0.1:8111/rest/\$catalog/getName

戻り値

```
{  
  "result": "US cities and zip codes manager"  
}
```

DataClassクラス関数を使用する

City DataClassクラスは、引数として受け取った名前をもとに City エンティティを返す API を提供しています:

```
// cs.City クラス  
  
Class extends DataClass  
  
exposed Function getCity()  
  var $0 : cs.CityEntity  
  var $1,$nameParam : text  
  $nameParam:=$1  
  $0:=This.query("name = :1";$nameParam).first()
```

次のリクエストを実行します:

POST 127.0.0.1:8111/rest/City/getCity

リクエストのボディ: ["Aguada"]

戻り値

結果は、次のエンティティです:

```
{  
  "__entityModel": "City",  
  "__DATACLASS": "City",  
  "__KEY": "1",  
  "__TIMESTAMP": "2020-03-09T08:03:19.923Z",  
  "__STAMP": 1,  
  "ID": 1,  
  "name": "Aguada",  
  "countyFIPS": 72003,  
  "county": {  
    "__deferred": {  
      "uri": "/rest/County(72003)",  
      "__KEY": "72003"  
    }  
  },  
  "zips": {  
    "__deferred": {  
      "uri": "/rest/City(1)/zips?$expand=zips"  
    }  
  }  
}
```

Entityクラス関数を使用する

CityEntity Entityクラスは API を提供しています:

```
// cs.CityEntity クラス  
  
Class extends Entity  
  
exposed Function getPopulation()  
$0:=This.zips.sum("population")
```

次のリクエストを実行します:

```
POST 127.0.0.1:8111/rest/City(2)/getPopulation
```

戻り値

```
{  
    "result": 48814  
}
```

EntitySelectionクラス関数を使用する

CitySelection EntitySelectionクラスは API を提供しています:

```
// cs.CitySelection クラス  
  
Class extends EntitySelection  
  
exposed Function getPopulation()  
$0:=This.zips.sum("population")
```

次のリクエストを実行します:

```
POST 127.0.0.1:8111/rest/City/getPopulation/?$filter="ID<3"
```

戻り値

```
{  
    "result": 87256  
}
```

EntitySelectionクラス関数とエンティティセットを使用する

StudentsSelection クラスは getAgeAverage 関数を持ちます:

```
// cs.StudentsSelection クラス  
  
Class extends EntitySelection  
  
exposed Function getAgeAverage  
C_LONGINT($sum;$0)  
C_OBJECT($s)  
  
$sum:=0  
For each ($s;This)  
    $sum:=$sum+$s.age()  
End for each  
$0:=$sum/This.length
```

あらかじめ作成した既存のエンティティセットを使い、次のリクエストを実行します：

```
POST 127.0.0.1:8044/rest/Students/getAgeAverage/$entityset/17E83633FFB54ECDBF947E5C620BB532
```

戻り値

```
{  
    "result": 34  
}
```

EntitySelectionクラス関数と orderBy を使用する

`StudentsSelection` クラスは `getLastSummary` 関数を持ちます：

```
// cs.StudentsSelection クラス  
  
Class extends EntitySelection  
  
exposed Function getLastSummary  
    C_TEXT($0)  
    C_OBJECT($last)  
  
    $last:=This.last()  
    $0:=$last.firstname + " - +" + $last.lastname + " is ... " + String($last.age())
```

次のリクエストを実行します：

```
POST 127.0.0.1:8044/rest/Students/getLastSummary/$entityset/?$filter=lastname=b@"&$orderby=lastname"
```

戻り値

```
{  
    "result": "Wilbert - Bull is ... 21"  
}
```

エンティティを作成する

`Students` DataClassクラスは、データを含むエンティティをクライアントから受け取る `pushData()` 関数を持ちます。 `checkData()` メソッドはいくつかの検証を実行します。問題がなければ、エンティティは保存されて返されます。

```
// cs.Students クラス

Class extends DataClass

exposed Function pushData
    var $1, $entity, $status, $0 : Object

    $entity:=$1

    $status:=checkData($entity) // $status は success ブールプロパティを持つオブジェクトです

    $0:=$status

    If ($status.success)
        $status:=$entity.save()
        If ($status.success)
            $0:=$entity
    End if
End if
```

次のリクエストを実行します:

POST <http://127.0.0.1:8044/rest/Students/pushData>

リクエストのボディ:

```
[{
    "__DATACLASS": "Students",
    "__ENTITY": true,
    "firstname": "Ann",
    "lastname": "Brown"
}]
```

__KEY が提供されていないため、サーバー上では クライアントから受け取った属性を持つ 新規の Studentsエンティティが読み込まれます。
pushData() 関数が save() を実行するため、この新規エンティティは保存されます。

戻り値

```
{
    "__entityModel": "Students",
    "__DATACLASS": "Students",
    "__KEY": "55",
    "__TIMESTAMP": "2020-06-16T10:54:41.805Z",
    "__STAMP": 1,
    "ID": 55,
    "firstname": "Ann",
    "lastname": "BROWN",
    "schoolID": null,
    "school": null
}
```

エンティティを更新する

__KEY 属性を使って、上の例題と同じことをおこなうと、エンティティを更新します。

次のリクエストを実行します:

POST: <http://127.0.0.1:8044/rest/Students/pushData>

リクエストのボディ:

```
[{  
    "__DATACLASS": "Students",  
    "__ENTITY": true,  
    "lastname": "Brownie",  
    "__KEY": 55  
}]
```

`__KEY` が提供されているため、クライアントから受け取った `lastname` 属性値を持つ プライマリーキーが 55 の Students エンティティが読み込まれます。pushData() 関数が `save()` を実行するため、このエンティティは更新されます。

戻り値

```
{  
    "__entityModel": "Students",  
    "__DATACLASS": "Students",  
    "__KEY": "55",  
    "__TIMESTAMP": "2020-06-16T11:10:21.679Z",  
    "__STAMP": 3,  
    "ID": 55,  
    "firstname": "Ann",  
    "lastname": "BROWNIE",  
    "schoolID": null,  
    "school": null  
}
```

リレートエンティティを持つエンティティを作成する

プライマリーキー 2 を持つ Schools エンティティをリレートエンティティとして、新規 Students エンティティを作成します。

次のリクエストを実行します:

POST: <http://127.0.0.1:8044/rest/Students/pushData>

リクエストのボディ:

```
[{  
    "__DATACLASS": "Students",  
    "__ENTITY": true,  
    "firstname": "John",  
    "lastname": "Smith",  
    "school": {"__KEY": 2}  
}]
```

戻り値

```
{
    "__entityModel": "Students",
    "__DATACLASS": "Students",
    "__KEY": "56",
    "__TIMESTAMP": "2020-06-16T11:16:47.601Z",
    "__STAMP": 1,
    "ID": 56,
    "firstname": "John",
    "lastname": "SMITH",
    "schoolID": 2,
    "school": {
        "__deferred": {
            "uri": "/rest/Schools(2)",
            "__KEY": "2"
        }
    }
}
```

リレートエンティティを持つエンティティを更新する

既存の Schools エンティティを既存の Students エンティティに紐付けます。 `StudentsEntity` クラスは次の API を提供しています:

```
// cs.StudentsEntity クラス

Class extends Entity

exposed Function putToSchool()
    var $1, $school, $0, $status : Object

    // $1 は Schools エンティティ
    $school:=$1
    // Schools リレートエンティティをカレントの Students エンティティに紐付けます
    This.school:=$school // このとき、school は N対1リレーション名です

    $status:=This.save()

    $0:=$status
```

Students エンティティを対象に次のリクエストを実行します:

POST `http://127.0.0.1:8044/rest/Students(1)/putToSchool`

リクエストのボディ:

```
[{
    "__DATACLASS": "Schools",
    "__ENTITY": true,
    "__KEY": 2
}]
```

戻り値

```
{
    "result": {
        "success": true
    }
}
```

エンティティセレクションを引数として受け取る

Students DataClassクラスは、受け取ったエンティティセレクション (\$1) を更新する `setFinalExam()` 関数を持ちます。実際には、エンティティセレクション内の各エンティティの `finalExam` 属性値を、2つ目に渡した引数 (\$2) に更新します。最後に、更新されたエンティティのプライマリーキーを返します。

```
// cs.Students クラス

Class extends DataClass

exposed Function setFinalExam()

var $1, $es, $student, $status : Object
var $2, $examResult : Text

var $keys, $0 : Collection

// エンティティセレクション
$es:=$1

$examResult:=$2

$keys:=New collection()

// エンティティセレクションをループします
For each ($student;$es)
    $student.finalExam:=$examResult
    $status:=$student.save()
    If ($status.success)
        $keys.push($student.ID)
    End if
End for each

$0:=$keys
```

次のようなリクエストでエンティティセットをあらかじめ作成します:

```
http://127.0.0.1:8044/rest/Students/?$filter="ID<3"&$method=entityset
```

次のリクエストを実行します:

```
POST http://127.0.0.1:8044/rest/Students/setFinalExam
```

リクエストのボディ:

```
[
{
  "__ENTITIES":true,
  "__DATASET":"9B9C053A111E4A288E9C1E48965FE671"
},
"Passed"
]
```

戻り値

プライマリーキー 1と2 のエンティティが更新されました:

```
{  
  "result": [  
    1,  
    2  
  ]  
}
```

クライアント側で更新されたエンティティセレクションを使用する

前述の `getAgeAverage()` 関数を使います。

```
var $remoteDS, $newStudent, $students : Object  
var $ageAverage : Integer  
  
$remoteDS:=open datastore(New object("hostname","127.0.0.1:8044");"students")  
  
// $newStudent は処理する Studentsエンティティです  
$newStudent:=...  
$students:=$remoteDS.Students.query("school.name = :1";"Math school")  
// クライアント側で $students エンティティセレクションにエンティティを追加します  
$students.add($newStudent)  
  
// StudentsSelectionクラスに対して、同セレクション内の生徒エンティティの平均年齢を返す関数を呼び出します  
// この関数は、クライアント側の追加エンティティを含む更新された内容の $students エンティティセレクションに対して、サーバー-$ageAverage:=$students.getAgeAverage()
```

RESTリクエストについて

RESTリクエストでは次の構文がサポートされています:

URI	リソース(入力)	/? または &{filter} (出力)
http://{servername}:{port}/rest/	{dataClass}	\$filter, \$attributes, \$skip, \$method=.....
	{dataClass}/\$entityset/{entitySetID}	\$method=...
	{dataClass}({key})	\$attributes
	{dataClass}:{attribute}(value)	

RESTリクエストには、URI とリソースが必ず含まれていなければなりませんが、返されるデータをフィルターする出力パラメーターの使用は任意です。

すべての URI と同様に、先頭パラメーターは "?" に続けて指定し、それ以降のパラメーターは "&" で区切れます。たとえば:

```
GET /rest/Person/?$filter="lastName!=Jones"&$method=entityset&$timeout=600
```

曖昧さ回避のため、値は引用符内に書くことができます。たとえば、上の例では名字 (lastName) の値を単一引用符内に書けます:
"lastName!='Jones'"。

パラメーターを利用することで、4Dプロジェクトのデータクラスのデータを操作できます。 GET HTTPメソッドを使ってデータを取得する以外にも、 POST HTTPメソッドを使ってデータクラスのエンティティを追加・更新・削除することができます。

JSON の代わりに配列形式でデータを取得するには `$asArray` パラメーターを使います。

RESTステータスとレスポンス

各 RESTリクエストに対し、サーバーはステータスとレスポンス (エラー付き、またはエラー無し) を返します。

リクエストステータス

RESTリクエストをおこなうと、レスポンスとともにステータスが返されます。主なステータスをいくつか紹介します:

ステータス	説明
0	リクエストは処理されませんでした (サーバー未起動の可能性)
200 OK	リクエストはエラーなく処理されました
401 Unauthorized	権限エラー (ユーザーのアクセス権限を確認する必要があります)
402 No session	セッションの最大数に達しています
404 Not Found	データクラスが REST に公開されていないか、エンティティが存在しません
500 Internal Server Error	RESTリクエスト処理中にエラーが発生しました

レスポンス

返されるレスポンス (JSON形式) はリクエストによって変わります。

エラーが発生した場合、その内容はレスポンスとともに返されるか、サーバーのレスポンスそのものになります。

\$catalog

カタログには、データストアを構成するすべてのデータクラスおよび属性の詳細な情報が含まれます。

使用可能なシンタックス

シンタックス	例題	説明
\$catalog	/\$catalog	プロジェクト内のデータクラスのリストを、2つの URI とともに返します。
\$catalog/\$all	/\$catalog/\$all	プロジェクト内のすべてのデータクラスとそれらの属性の情報を返します。
\$catalog/{dataClass}	/\$catalog/Employee	特定のデータクラスとその属性の情報を返します。

\$catalog

プロジェクト内のデータクラスのリストを、2つの URI とともに返します。1つはデータクラスのストラクチャー情報にアクセスするためのもので、もう1つはデータクラスのデータを取得するためのものです。

説明

\$catalog を呼び出すと、プロジェクトのデータストア内のデータクラスのリストを、データクラス毎に 2つの URI とともに返します。

プロジェクトのデータストア内の、公開されているデータクラスのみがリストされます。詳細については、テーブルやフィールドの公開** を参照してください。

データクラス毎に返されるプロパティの説明です：

プロパティ	タイプ	説明
name	String	データクラスの名称。
uri	String	データクラスとその属性に関する情報を取得するための URI です。
dataURI	String	データクラスのデータを取得するための URI です。

例題

GET /rest/\$catalog

結果：

```
{
  dataClasses: [
    {
      name: "Company",
      uri: "http://127.0.0.1:8081/rest/$catalog/Company",
      dataURI: "http://127.0.0.1:8081/rest/Company"
    },
    {
      name: "Employee",
      uri: "http://127.0.0.1:8081/rest/$catalog/Employee",
      dataURI: "http://127.0.0.1:8081/rest/Employee"
    }
  ]
}
```

\$catalog/\$all

プロジェクト内のすべてのデータクラスとそれらの属性の情報を返します。

説明

`$catalog/$all` を呼び出すと、プロジェクトのデータストア内の各データクラスについて属性の情報を取得します。

各データクラスと属性について取得される情報についての詳細は [\\$catalog/{dataClass}](#) を参照ください。

例題

GET /rest/\$catalog/\$all

結果:

```
{  
  "dataClasses": [  
    {  
      "name": "Company",  
      "className": "Company",  
      "collectionName": "CompanySelection",  
      "tableNumber": 2,  
      "scope": "public",  
      "dataURI": "/rest/Company",  
      "attributes": [  
        {  
          "name": "ID",  
          "kind": "storage",  
          "fieldPos": 1,  
          "scope": "public",  
          "indexed": true,  
          "type": "long",  
          "identifying": true  
        },  
        {  
          "name": "name",  
          "kind": "storage",  
          "fieldPos": 2,  
          "scope": "public",  
          "type": "string"  
        },  
        {  
          "name": "revenues",  
          "kind": "storage",  
          "fieldPos": 3,  
          "scope": "public",  
          "type": "number"  
        },  
        {  
          "name": "staff",  
          "kind": "relatedEntities",  
          "fieldPos": 4,  
          "scope": "public",  
          "type": "EmployeeSelection",  
          "reversePath": true,  
          "path": "employer"  
        },  
        {  
          "name": "url",  
          "kind": "storage",  
          "scope": "public",  
          "type": "string"  
        }  
      ]  
    }  
  ]  
}
```

```

        }
    ],
    "key": [
        {
            "name": "ID"
        }
    ]
},
{
    "name": "Employee",
    "className": "Employee",
    "collectionName": "EmployeeSelection",
    "tableNumber": 1,
    "scope": "public",
    "dataURI": "/rest/Employee",
    "attributes": [
        {
            "name": "ID",
            "kind": "storage",
            "scope": "public",
            "indexed": true,
            "type": "long",
            "identifying": true
        },
        {
            "name": "firstname",
            "kind": "storage",
            "scope": "public",
            "type": "string"
        },
        {
            "name": "lastname",
            "kind": "storage",
            "scope": "public",
            "type": "string"
        },
        {
            "name": "employer",
            "kind": "relatedEntity",
            "scope": "public",
            "type": "Company",
            "path": "Company"
        }
    ],
    "key": [
        {
            "name": "ID"
        }
    ]
}
]
}

```

\$catalog/{dataClass}

特定のデータクラスとその属性の情報を返します。

説明

`$catalog/{dataClass}` を呼び出すと、指定したデータクラスとその属性について詳細な情報が返されます。プロジェクトのデータストア内のすべてのデータクラスに関して同様の情報を得るには `$catalog/$all` を使います。

返される情報は次の通りです:

- データクラス
- 属性
- メソッド (あれば)
- プライマリーキー

データクラス

公開されているデータクラスについて、次のプロパティが返されます:

プロパティ	タイプ	説明
name	String	データクラスの名称
collectionName	String	データクラスにおいて作成されるエンティティセレクションの名称
tableNumber	数値	4Dデータベース内のテーブル番号
scope	String	データクラスのスコープ (公開 (public) に設定されているデータクラスのみ返されます)
dataURI	String	データクラスのデータを取得するための URI

属性

公開されている各属性について、次のプロパティが返されます:

プロパティ	タイプ	説明
name	String	属性の名称
kind	String	属性タイプ (ストレージ (storage) またはリレートエンティティ (relatedEntity))
fieldPos	数値	データベーステーブルのフィールド番号
scope	String	属性のスコープ (公開 (public) に設定されている属性のみ返されます)
indexed	String	属性にインデックスが設定されていれば、このプロパティは true を返します。それ以外の場合には、このプロパティは表示されません。
type	String	属性タイプ (bool, blob, byte, date, duration, image, long, long64, number, string, uuid, word)、または、N->1 リレーション属性の場合はリレーション先のデータクラス
identifying	ブール	属性がプライマリーキーの場合、プロパティは true を返します。それ以外の場合には、このプロパティは表示されません。
path	String	relatedEntity 属性の場合はデータクラス名、relatedEntities 属性の場合はリレーション名
foreignKey	String	relatedEntity 属性の場合、リレート先の属性名
inverseName	String	relatedEntity または relatedEntities 属性の逆方向リレーション名

プライマリーキー

key オブジェクトには、データクラスの プライマリーキー として定義された属性の 名称 (name プロパティ) が返されます。

例題

特定のデータクラスに関する情報を取得します。

```
GET /rest/$catalog/Employee
```

結果:

```
{
  name: "Employee",
  className: "Employee".
```

```
        ],
        collectionName: "EmployeeCollection",
        scope: "public",
        dataURI: "http://127.0.0.1:8081/rest/Employee",
        defaultTopSize: 20,
        extraProperties: {
            panelColor: "#76923C",
            __CDATA: "\n\n\t\t\n",
            panel: {
                isOpen: "true",
                pathVisible: "true",
                __CDATA: "\n\n\t\t\t\n",
                position: {
                    X: "394",
                    Y: "42"
                }
            }
        },
        attributes: [
            {
                name: "ID",
                kind: "storage",
                scope: "public",
                indexed: true,
                type: "long",
                identifying: true
            },
            {
                name: "firstName",
                kind: "storage",
                scope: "public",
                type: "string"
            },
            {
                name: "lastName",
                kind: "storage",
                scope: "public",
                type: "string"
            },
            {
                name: "fullName",
                kind: "calculated",
                scope: "public",
                type: "string",
                readOnly: true
            },
            {
                name: "salary",
                kind: "storage",
                scope: "public",
                type: "number",
                defaultFormat: {
                    format: "$###,###.00"
                }
            },
            {
                name: "photo",
                kind: "storage",
                scope: "public",
                type: "image"
            },
            {
                name: "employer",
                kind: "relatedEntity",
                scope: "public",
                type: "Company",
                extraProperties: {
                    panelColor: "#76923C",
                    __CDATA: "\n\n\t\t\n",
                    panel: {
                        isOpen: "true",
                        pathVisible: "true",
                        __CDATA: "\n\n\t\t\t\n",
                        position: {
                            X: "394",
                            Y: "42"
                        }
                    }
                }
            }
        ]
    }
}
```

```
        path: "Company"
    },
{
    name: "employerName",
    kind: "alias",
    scope: "public",

    type: "string",
    path: "employer.name",
    readOnly: true
},
{
    name: "description",
    kind: "storage",
    scope: "public",
    type: "string",
    multiLine: true
},
],
key: [
{
    name: "ID"
}
]
}
```

\$directory

ディレクトリは RESTリクエストを介したユーザーアクセスに対応します。

\$directory/login

4Dアプリケーション上で RESTセッションを開き、ユーザーをログインします。

説明

RESTを介して 4Dアプリケーション上でセッションを開き、ユーザーをログインするには、`$directory/login` を使用します。デフォルトの 4Dセッションタイムアウトを変更することもできます。

パラメーターはすべて、POST の ヘッダーに渡す必要があります：

ヘッダーキー	ヘッダー値
username-4D	ユーザー (任意)
password-4D	パスワード (任意)
hashed-password-4D	ハッシュ化パスワード (任意)
session-4D-length	セッション非アクティブタイムアウト (分単位)。60 以上の値 (任意)

例題

```
C_TEXT($response;$body_t)
ARRAY TEXT($hKey;3)
ARRAY TEXT($hValues;3)
$hKey{1}:="username-4D"
$hKey{2}:="hashed-password-4D"
$hKey{3}:="session-4D-length"
$hValues{1}:="john"
$hValues{2}:=Generate digest("123";4D digest)
$hValues{3}:=120
$httpStatus:=HTTP Request(HTTP POST method;"app.example.com:9000/rest/$directory/login";$body_t;$response
```

結果:

ログインに成功した場合の結果:

```
{
  "result": true
}
```

それ以外の場合の結果:

```
{
  "result": false
}
```

\$info

4D Server のキャッシュに保存されているエンティティセットおよびユーザーセッションの情報を返します。

説明

プロジェクトに対してこのリクエストを送信すると、次のプロパティに情報を取得します：

プロパティ	タイプ	説明
cacheSize	数値	4D Server のキャッシュサイズ
usedCache	数値	4D Server のキャッシュ使用量
entitySetCount	数値	エンティティセットの数
entitySet	コレクション	各エンティティセットの情報が格納されているオブジェクトのコレクション
ProgressInfo	コレクション	進捗インジケーターの情報が格納されているコレクション
sessionInfo	コレクション	各ユーザーセッションの情報が格納されているオブジェクトのコレクション

entitySet

4D Server のキャッシュに保存されている各エンティティセットについて、次の情報が返されます：

プロパティ	タイプ	説明
id	String	エンティティセットを参照する UUID
dataClass	String	データクラスの名称。
selectionSize	数値	エンティティセットに含まれるエンティティの数
sorted	布尔	エンティティセットが (\$orderby の使用により) 順列ありの場合には true、順列なしの場合は false。
refreshed	日付	エンティティセットが最後に使用された日付または作成日。
expires	日付	エンティティセットの有効期限 (エンティティセットが更新されるたびに、この日付/時間は変更されます)。 expires と refreshed の差がエンティティセットのタイムアウトです。デフォルトのタイムアウトは2時間ですが、 \$timeout を使って指定することもできます。

エンティティセットを作成する方法についての詳細は `$method=entityset` を参照ください。4D Server のキャッシュからエンティティセットを削除したい場合には `$method=release` を使います。

最適化のため、4D は独自のエンティティセットを生成します。つまり、`$method=entityset` で作成した以外のエンティティセットも返されます。重要 プロジェクトにおいて、4D の パスワードアクセスシステムを起動している場合には、Adminグループのユーザーとしてログインしている必要があります。

sessionInfo

各ユーザーセッションについては、次の情報が `sessionInfo` コレクションに返されます：

プロパティ	タイプ	説明
sessionID	String	セッションを参照する UUID
userName	String	セッションを実行中のユーザー名
lifeTime	数値	ユーザー セッションのタイムアウト (デフォルトは 3600)
expiration	日付	ユーザー セッションの有効期限

例題

4D Server のキャッシュに保存されているエンティティセットおよびユーザー セッションの情報を取得します。

```
GET /rest/$info
```

結果:

```

{
cacheSize: 209715200,
usedCache: 3136000,
entitySetCount: 4,
entitySet: [
{
id: "1418741678864021B56F8C6D77F2FC06",
tableName: "Company",
selectionSize: 1,
sorted: false,
refreshed: "2011-11-18T10:30:30Z",
expires: "2011-11-18T10:35:30Z"
},
{
id: "CAD79E5BF339462E85DA613754C05CC0",
tableName: "People",
selectionSize: 49,
sorted: true,
refreshed: "2011-11-18T10:28:43Z",
expires: "2011-11-18T10:38:43Z"
},
{
id: "F4514C59D6B642099764C15D2BF51624",
tableName: "People",
selectionSize: 37,
sorted: false,
refreshed: "2011-11-18T10:24:24Z",
expires: "2011-11-18T12:24:24Z"
}
],
ProgressInfo: [
{
UserInfo: "flushProgressIndicator",
sessions: 0,
percent: 0
},
{
UserInfo: "indexProgressIndicator",
sessions: 0,
percent: 0
}
],
sessionInfo: [
{
sessionID: "6657ABBCEE7C3B4089C20D8995851E30",
userID: "36713176D42DB045B01B8E650E8FA9C6",
userName: "james",
lifeTime: 3600,
expiration: "2013-04-22T12:45:08Z"
},
{
sessionID: "A85F253EDE90CA458940337BE2939F6F",
userID: "00000000000000000000000000000000",
userName: "default guest",
lifeTime: 3600,
expiration: "2013-04-23T10:30:25Z"
}
]
}

```

エンティティセットに続く進捗インジケーターの情報は、4Dによって内部的に使用されます。

\$upload

サーバーにアップロードしたファイルの ID を返します

説明

サーバーにアップロードしたいファイルがある場合にこのリクエストを POST します。画像の場合には `$rawPict=true` を渡します。その他のファイルの場合は `$binary=true` を渡します。

デフォルトのタイムアウトは 120秒ですが、`$timeout` パラメーターに任意の数値を渡してタイムアウトを変更できます。

アップロードシナリオ

エンティティのピクチャー属性を更新するために、画像をアップロードしたい場合を考えます。

画像（または任意のバイナリファイル）をアップロードするには、まずクライアントアプリケーションにてファイルを選択する必要があります。ファイル自体はリクエストのボディに渡す必要があります。

次に、下のようなリクエストを使用して、選択した画像を 4D Server にアップロードします：

```
POST /rest/$upload?$rawPict=true
```

その結果、サーバーからはファイルを識別する ID が返されます。

レスポンス:

```
{ "ID": "D507BC03E613487E9B4C2F6A0512FE50" }
```

この画像をエンティティに追加するには、返された ID を使い `$method=update` で画像属性に保存します。リクエストは次のようになります：

```
POST /rest/Employee/?$method=update
```

POST データ:

```
{
  __KEY: "12",
  __STAMP: 4,
  photo: { "ID": "D507BC03E613487E9B4C2F6A0512FE50" }
}
```

レスポンス:

更新後のエンティティが返されます：

```
{
    "__KEY": "12",
    "__STAMP": 5,
    "uri": "http://127.0.0.1:8081/rest/Employee(12)",
    "ID": 12,
    "firstName": "John",
    "firstName": "Smith",
    "photo":
    {
        "__deferred":
        {
            "uri": "/rest/Employee(12)/photo?$imageformat=best&$version=1&$expand=photo",
            "image": true
        }
    }
},}
```

4D HTTPクライアントを使った例

次の例では、4D HTTPクライアントを使用して、.pdfファイルをサーバーにアップロードする方法を示します。

```
var $params : Text
var $response : Object
var $result : Integer
var $blob : Blob

ARRAY TEXT($headerNames; 1)
ARRAY TEXT($headerValues; 1)

$url:="localhost:80/rest/$upload?$binary=true" // RESTリクエストの準備

$headerNames{1}:="Content-Type"
$headerValues{1}:="application/octet-stream"

DOCUMENT TO BLOB("c:\\invoices\\inv003.pdf"; $blob) // バイナリの読み込み

// ファイルをアップロードするための 1つ目の POSTリクエスト
$result:=HTTP Request(HTTP POST method; $url; $blob; $response; $headerNames; $headerValues)

If ($result=200)
    var $data : Object
    $data:=New object
    $data.__KEY:="3"
    $data.__STAMP:="3"
    $data.pdf:=New object("ID"; String($response.ID))

    $url:="localhost:80/rest/Invoices?$method=update" // エンティティを更新するための 2つ目のリクエスト

    $headerNames{1}:="Content-Type"
    $headerValues{1}:="application/json"

    $result:=HTTP Request(HTTP POST method; $url; $data; $response; $headerNames; $headerValues)
Else
    ALERT(String($result)+" Error")
End if
```

{dataClass}

エンティティやセンティティセレクション、またはクラス関数を利用するにあたって、RESTリクエスト内にデータクラス名を直接使用することができます。

使用可能なシンタックス

シンタックス	例題	説明
{dataClass}	/Employee	データクラスの全データ (デフォルトでは先頭の 100 エンティティ) を返します
{dataClass}({key})	/Employee(22)	データクラスのプライマリーキーによって特定されるエンティティのデータを返します
{dataClass}:{attribute}(value)	/Employee:firstName(John)	指定した属性値を持つ 1 件のエンティティのデータを返します
{dataClass}/{DataClassClassFunction}	/City/getCity	DataClassクラス関数を実行します
{dataClass}({EntitySelectionClassFunction})	/City/getPopulation/?\$filter="ID<3"	EntitySelectionクラス関数を実行します
{dataClass}({key})/{EntityClassFunction}	City(2)/getPopulation	Entityクラス関数を実行します

関数の呼び出しについての詳細は [ORDAクラス関数](#) を参照ください。

{dataClass}

特定のデータクラス (例: `Company`) の全データ (デフォルトでは先頭の 100 エンティティ) を返します。

説明

RESTリクエストにこのパラメーターのみを渡すと、(

`$top/$limit` を使って指定しない限り) デフォルトで先頭の 100 件のエンティティが返されます。

返されるデータの説明です:

プロパティ	タイプ	説明
<code>__entityModel</code>	String	データクラスの名称。
<code>__COUNT</code>	数値	データクラスに含まれる全エンティティ数
<code>__SENT</code>	数値	RESTリクエストが返すエンティティの数。総エンティティ数が <code>\$top/\$limit</code> で指定された数より少なければ、総エンティティの数になります。
<code>__FIRST</code>	数値	セレクションの先頭エンティティの番号。デフォルトでは 0; または <code>\$skip</code> で指定された値。
<code>__ENTITIES</code>	コレクション	エンティティ毎にその属性をすべて格納したオブジェクトのコレクションです。リレーション属性は、リレーション先の情報取得するための URI を格納したオブジェクトとして返されます。

各エンティティには次のプロパティが含まれます:

プロパティ	タイプ	説明
__KEY	String	データクラスにおいて定義されているプライマリーキーの値
__TIMESTAMP	日付	エンティティが最後に編集された日時を記録するタイムスタンプ
__STAMP	数値	\$method=update を使ってエンティティの属性値を更新するときに必要となる内部スタンプ

取得する属性を指定するには、次のシンタックスを使っておこないます: `{attribute1, attribute2, ...}`。たとえば:

```
GET /rest/Company/name,address
```

例題

特定のデータクラスの全データを取得します。

```
GET /rest/Company
```

結果:

```
{
    "__entityModel": "Company",
    "__GlobalStamp": 51,
    "__COUNT": 250,
    "__SENT": 100,
    "__FIRST": 0,
    "__ENTITIES": [
        {
            "__KEY": "1",
            "__TIMESTAMP": "2020-04-10T10:44:49.927Z",
            "__STAMP": 1,
            "ID": 1,
            "name": "Adobe",
            "address": null,
            "city": "San Jose",
            "country": "USA",
            "revenues": 500000,
            "staff": {
                "__deferred": {
                    "uri": "http://127.0.0.1:8081/rest/Company(1)/staff?$expand=staff"
                }
            }
        },
        {
            "__KEY": "2",
            "__TIMESTAMP": "2018-04-25T14:42:18.351Z",
            "__STAMP": 1,
            "ID": 2,
            "name": "Apple",
            "address": null,
            "city": "Cupertino",
            "country": "USA",
            "revenues": 890000,
            "staff": {
                "__deferred": {
                    "uri": "http://127.0.0.1:8081/rest/Company(2)/staff?$expand=staff"
                }
            }
        },
        {
            "__KEY": "3",
            "__TIMESTAMP": "2018-04-23T09:03:49.021Z",
            "__STAMP": 2,
            "ID": 3,
            "name": "4D",
            "address": null,
            "city": "Paris"
        }
    ]
}
```

```

    "address": null,
    "city": "Clichy",
    "country": "France",
    "revenues": 700000,
    "staff": {
        "__deferred": {
            "uri": "http://127.0.0.1:8081/rest/Company(3)/staff?$expand=staff"
        }
    }
},
{
    "__KEY": "4",
    "__TIMESTAMP": "2018-03-28T14:38:07.430Z",
    "__STAMP": 1,
    "ID": 4,
    "name": "Microsoft",
    "address": null,
    "city": "Seattle",
    "country": "USA",
    "revenues": 650000,
    "staff": {
        "__deferred": {
            "uri": "http://127.0.0.1:8081/rest/Company(4)/staff?$expand=staff"
        }
    }
}
....//more entities here
]
}

```

{dataClass}({key})

データクラスのプライマリーキーによって特定されるエンティティのデータを返します（例: `Company(22)` または `Company("IT0911AB2200")` など）。

説明

データクラスとキーを渡すことで、公開されているエンティティの情報を取得することができます。キー (key) は、データクラスに定義されているプライマリーキーの値です。プライマリーキーの定義についての詳細は、デザインリファレンスマニュアルの [主キーを設定、削除する](#) を参照ください。

返されるデータについての詳細は [{DataClass}](#) を参照ください。

取得する属性を指定するには、次のシンタックスを使っておこないます: `{attribute1, attribute2, ...}`。たとえば:

```
GET /rest/Company(1)/name,address
```

`$expand` を使ってリレーション属性を展開するには、次のように指示します:

```
GET /rest/Company(1)/name,address,staff?$expand=staff
```

例題

次のリクエストは、Company データクラスで主キーが 1 であるエンティティの公開データをすべて返します。

```
GET /rest/Company(1)
```

結果:

```
{  
    "__entityModel": "Company",  
    "__KEY": "1",  
    "__TIMESTAMP": "2020-04-10T10:44:49.927Z",  
    "__STAMP": 2,  
    "ID": 1,  
    "name": "Apple",  
    "address": Infinite Loop,  
    "city": "Cupertino",  
    "country": "USA",  
    "url": http://www.apple.com,  
    "revenues": 500000,  
    "staff": {  
        "__deferred": {  
            "uri": "http://127.0.0.1:8081/rest/Company(1)/staff?$expand=staff"  
        }  
    }  
}
```

{dataClass}:{attribute}(value)

指定した属性値を持つ 1件のエンティティのデータを返します

説明

dataClass に加えて attribute (属性) および value (値)を渡すことで、当該エンティティの公開データをすべて取得できます。指定する値は、その属性において一意のものですが、主キーではありません。

```
GET /rest/Company:companyCode(Acme001)
```

取得する属性を指定するには、次のシンタクスを使っておこないます: {attribute1, attribute2, ...}。たとえば:

```
GET /rest/Company:companyCode(Acme001)/name,address
```

\$attributes を使ってリレーション属性を使用するには、次のように指示します:

```
GET /rest/Company:companyCode(Acme001)?$attributes=name,address,staff.name
```

例題

次のリクエストは、名前が "Jones" である社員 (Employee) の公開データをすべて返します。

```
GET /rest/Employee:lastname(Jones)
```

\$asArray

クエリの結果を、JSONオブジェクトではなく配列（コレクション）として返します。

説明

レスポンスを配列として取得するには、RESTリクエストに `$asArray` を追加します（例： `$asArray=true`）。

例題

配列としてレスポンスを取得する例です。

```
GET /rest/Company/?$filter=name begin a"&$top=3&$asArray=true
```

レスポンス：

```
[  
  {  
    "__KEY": 15,  
    "__STAMP": 0,  
    "ID": 15,  
    "name": "Alpha North Yellow",  
    "creationDate": "!!0000-00-00!!",  
    "revenues": 82000000,  
    "extra": null,  
    "comments": "",  
    "__GlobalStamp": 0  
  },  
  {  
    "__KEY": 34,  
    "__STAMP": 0,  
    "ID": 34,  
    "name": "Astral Partner November",  
    "creationDate": "!!0000-00-00!!",  
    "revenues": 90000000,  
    "extra": null,  
    "comments": "",  
    "__GlobalStamp": 0  
  },  
  {  
    "__KEY": 47,  
    "__STAMP": 0,  
    "ID": 47,  
    "name": "Audio Production Uniform",  
    "creationDate": "!!0000-00-00!!",  
    "revenues": 28000000,  
    "extra": null,  
    "comments": "",  
    "__GlobalStamp": 0  
  }  
]
```

同じデータをデフォルトの JSON形式で取得した場合です：

```
{
  "__entityModel": "Company",
  "__GlobalStamp": 50,
  "__COUNT": 52,
  "__FIRST": 0,
  "__ENTITIES": [
    {
      "__KEY": "15",
      "__TIMESTAMP": "2018-03-28T14:38:07.434Z",
      "__STAMP": 0,
      "ID": 15,
      "name": "Alpha North Yellow",
      "creationDate": "0!0!0",
      "revenues": 82000000,
      "extra": null,
      "comments": "",
      "__GlobalStamp": 0,
      "employees": {
        "__deferred": {
          "uri": "/rest/Company(15)/employees?$expand=employees"
        }
      }
    },
    {
      "__KEY": "34",
      "__TIMESTAMP": "2018-03-28T14:38:07.439Z",
      "__STAMP": 0,
      "ID": 34,
      "name": "Astral Partner November",
      "creationDate": "0!0!0",
      "revenues": 90000000,
      "extra": null,
      "comments": "",
      "__GlobalStamp": 0,
      "employees": {
        "__deferred": {
          "uri": "/rest/Company(34)/employees?$expand=employees"
        }
      }
    },
    {
      "__KEY": "47",
      "__TIMESTAMP": "2018-03-28T14:38:07.443Z",
      "__STAMP": 0,
      "ID": 47,
      "name": "Audio Production Uniform",
      "creationDate": "0!0!0",
      "revenues": 28000000,
      "extra": null,
      "comments": "",
      "__GlobalStamp": 0,
      "employees": {
        "__deferred": {
          "uri": "/rest/Company(47)/employees?$expand=employees"
        }
      }
    }
  ],
  "__SENT": 3
}
```

\$atomic/\$atonce

RESTリクエストに含まれる操作をトランザクション内で処理します。エラーがなかった場合、トランザクションは受け入れられます。それ以外の場合、トランザクションはキャンセルされます。

説明

複数の操作を一回のリクエストで処理する際には `$atomic/$atonce` を使うことで、1つでも操作に問題があった場合にすべての操作をキャンセルすることができます。`$atomic` および `$atonce` のどちらでも利用できます。

例題

次の RESTリクエストをトランザクション内で呼び出します。

```
POST /rest/Employee?method=update&$atomic=true
```

POST データ:

```
[  
 {  
   "__KEY": "200",  
   "firstname": "John"  
 },  
 {  
   "__KEY": "201",  
   "firstname": "Harry"  
 }  
 ]
```

2つ目のエンティティの操作中に次のエラーが発生します。そのため、1つ目のエンティティも保存されません:

```
{
    "__STATUS": {
        "success": true
    },
    "__KEY": "200",
    "__STAMP": 1,
    "uri": "/rest/Employee(200)",
    "__TIMESTAMP": "!!2020-04-03!!",
    "ID": 200,
    "firstname": "John",
    "lastname": "Keeling",
    "isWoman": false,
    "numberOfKids": 2,
    "addressID": 200,
    "gender": false,
    "address": {
        "__deferred": {
            "uri": "/rest/Address(200)",
            "__KEY": "200"
        }
    },
    "__ERROR": [
        {
            "message": "Cannot find entity with \"201\" key in the \"Employee\" dataclass",
            "componentSignature": "dbmg",
            "errCode": 1542
        }
    ]
}
```

1つ目のエンティティの名前は "John" ですが、この値はサーバー上に保存されず、タイムスタンプも変更されません。したがって、エンティティをリードすると、もとの値に戻ります。

\$attributes

データクラスから取得するリレート属性を選択するのに使います (例: `Company(1)?$attributes=employees.lastname`、`Employee? $attributes=employer.name`)。

説明

データクラスにリレーション属性が含まれていて、リレート先のエンティティまたはエンティティセレクションの属性のうち値を取得するものを選択したい場合、そのパスを指定するのに `$attributes` を使用します。

`$attributes` はエンティティ (例: `People(1)`) またはエンティティセレクション (例: `People/$entityset/0AF4679A5C394746BFEB68D2162A19FF`) に対して適用できます。

- クエリに `$attributes` が指定されていない場合、または "*" が渡された場合、すべての取得可能な属性が取得されます。リレートエンティティ属性は、`__KEY` (プライマリーキー) と `URI` プロパティを持つオブジェクトという簡単な形で抽出されます。リレートエンティティズ 属性は抽出されません。
- リレートエンティティ 属性を対象に `$attributes` が指定された場合:
 - `$attributes=relatedEntity` : リレートエンティティは簡単な形で返されます (`__KEY` (プライマリーキー) と `URI` プロパティを持つ deferred オブジェクト)
 - `$attributes=relatedEntity.*` : リレートエンティティの属性がすべて返されます。
 - `$attributes=relatedEntity.attributePath1, relatedEntity.attributePath2, ...` : リレートエンティティの指定された属性だけが返されます。
- リレートエンティティズ 属性を対象に `$attributes` が指定された場合:
 - `$attributes=relatedEntities.*` : リレートエンティティズの属性がすべて返されます。
 - `$attributes=relatedEntities.attributePath1, relatedEntities.attributePath2, ...` : リレートエンティティズの指定された属性だけが返されます。

リレートエンティティズの例

"employees" 1対Nリレーションを持つ Company データクラスに対して次の RESTリクエストをおこなうと:

```
GET /rest/Company(1)/?$attributes=employees.lastname
```

レスポンス:

```
{
    "__entityModel": "Company",
    "__KEY": "1",
    "__TIMESTAMP": "2018-04-25T14:41:16.237Z",
    "__STAMP": 2,
    "employees": {
        "__ENTITYSET": "/rest/Company(1)/employees?$expand=employees",
        "__GlobalStamp": 50,
        "__COUNT": 135,
        "__FIRST": 0,
        "__ENTITIES": [
            {
                "__KEY": "1",
                "__TIMESTAMP": "2019-12-01T20:18:26.046Z",
                "__STAMP": 5,
                "lastname": "ESSEAL"
            },
            {
                "__KEY": "2",
                "__TIMESTAMP": "2019-12-04T10:58:42.542Z",
                "__STAMP": 6,
                "lastname": "JONES"
            },
            ...
        ]
    }
}
```

employees の属性をすべて取得するには:

```
GET /rest/Company(1)/?$attributes=employees.*
```

また、employees の lastname 属性と jobname 属性を取得するには:

```
GET /rest/Company(1)/?$attributes=employees.lastname,employees.jobname
```

リレートエンティティの例

"employer" N対1リレーションを持つ Employee データクラスに対して次の RESTリクエストをおこなう:

```
GET /rest/Employee(1)?$attributes=employer.name
```

レスポンス:

```
{
    "__entityModel": "Employee",
    "__KEY": "1",
    "__TIMESTAMP": "2019-12-01T20:18:26.046Z",
    "__STAMP": 5,
    "employer": {
        "__KEY": "1",
        "__TIMESTAMP": "2018-04-25T14:41:16.237Z",
        "__STAMP": 0,
        "name": "Adobe"
    }
}
```

employer の属性をすべて取得するには:

```
GET /rest/Employee(1)?$attributes=employer.*
```

また、employer の全 employees の lastname 属性を取得するには:

```
GET /rest/Employee(1)?$attributes=employer.employees.lastname
```

\$binary

ドキュメントを BLOB として保存するには "true" を渡します (`$expand={blobAttributeName}` も渡す必要があります)

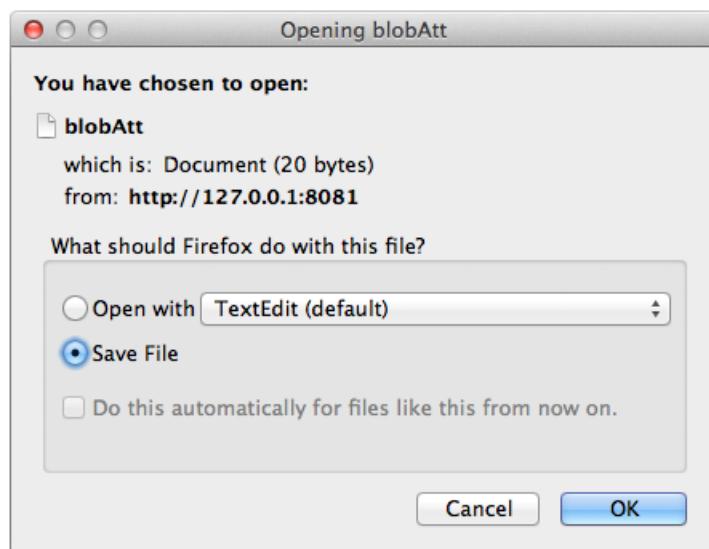
説明

`$binary` を使うと、ドキュメントを BLOB として保存できます。 `$expand` コマンドとの組み合わせで使う必要があります。

以下のリクエストを実行した場合:

```
GET /rest/Company(11)/blobAtt?$binary=true&$expand=blobAtt
```

ディスク上の BLOB の保存先を聞かれます:



\$compute

指定した属性を対象に計算をおこないます (例: Employee/salary/?\$compute=sum。オブジェクト属性の例: Employee/objectAtt.property1/?\$compute=sum)。

説明

このパラメーターを使って、データを対象に計算をおこなうことができます。

属性に対して計算をおこなうには、次のように書きます:

```
GET /rest/Employee/salary/?$compute=$all
```

オブジェクト属性の場合は、プロパティを指定します。たとえば:

```
GET /rest/Employee/objectAtt.property1/?$compute=$all
```

次のキーワードが利用可能です:

キーワード	説明
\$all	利用可能なすべての計算を属性に対しておこない、結果を格納した JSON オブジェクトを取得します。数値型の属性については平均 (average)、カウント (count)、最小 (min)、最大 (max)、合計 (sum)、文字列型の属性についてはカウント (count)、最小 (min)、最大 (max) が利用可能です。
average	数値型属性の平均を取得します。
count	コレクション内の要素数またはデータクラス内のエンティティ数を取得します (どちらの場合も属性を指定する必要があります)
min	数値型属性あるいは文字列型属性の最小値を取得します。
max	数値型属性あるいは文字列型属性の最大値を取得します。
sum	数値型属性の合計を取得します。

例題

数値型の属性を対象にすべての計算値を取得するには、次のように書きます:

```
GET /rest/Employee/salary/?$compute=$all
```

レスポンス:

```
{
  "salary": {
    "count": 4,
    "sum": 335000,
    "average": 83750,
    "min": 70000,
    "max": 99000
  }
}
```

文字列型の属性を対象にすべての計算値を取得するには、次のように書きます:

```
GET /rest/Employee/firstName/?$compute=$all
```

レスポンス:

```
{  
    "salary": {  
        "count": 4,  
        "min": Anne,  
        "max": Victor  
    }  
}
```

属性に対して特定の計算のみをおこなうには、次のように書きます:

```
GET /rest/Employee/salary/?$compute=sum
```

レスポンス:

```
235000
```

オブジェクト属性に対して特定の計算のみをおこなうには、次のように書きます:

```
GET /rest/Employee/objectAttribute.property1/?$compute=sum
```

レスポンス:

```
45
```

\$distinct

指定した属性について、重複しない値のコレクションを取得します（例： Company/name?\$filter="name=a*"&\$distinct=true）

説明

`$distinct` を使って、指定した属性における重複しない値を格納したコレクションを取得することができます。その際、データクラスの属性を一つのみを指定することができます。通常は文字列型の属性を対象に使用しますが、複数の値を持つ属性であれば、その型に制限はありません。

対象となる要素を制限するのに `$skip` および `$top/$limit` も組み合わせて使用することができます。

例題

"a" で始まる会社名について、重複しない値のコレクションを取得するには、次のように書きます：

```
GET /rest/Company/name?$filter="name=a*"&$distinct=true
```

レスポンス：

```
[  
    "Adobe",  
    "Apple"  
]
```

\$entityset

`$method=entityset` を使って [エンティティセットを作成](#) すると、それを後で再利用することができます。

使用可能なシンタックス

シンタックス	例題	説明
<code>\$entityset/{entitySetID}</code>	<code>/People/\$entityset/0ANUMBER</code>	既存のエンティティセットを取得します
<code>\$entityset/{entitySetID}?\$operator...&\$otherCollection</code>	<code>/Employee/\$entityset/0ANUMBER? \$logicOperator=AND &\$otherCollection=C0ANUMBER</code>	既存エンティティセットの比較から新規エンティティセットを作成します

\$entityset/{entitySetID}

既存のエンティティセットを取得します(例: `People/$entityset/0AF4679A5C394746BFEB68D2162A19FF`)

説明

このシンタックスを使って、定義されたエンティティセットに対してあらゆる操作を実行できます。

エンティティセットには(デフォルトの、または `$timeout` で指定した) タイムリミットが設定されるため、`$savedfilter` や `$savedorderby` を使って、エンティティセットを作成する際に使用したフィルターや並べ替えの詳細を保存しておくこともできます。

4D Server のキャッシュに保存された既存のエンティティセットを取得する際に、次のいずれもエンティティセットに適用することができます: `$expand`, `$filter`, `$orderby`, `$skip`, `$top/$limit`。

例題

エンティティセットを作成すると、データとともにエンティティセットIDが返されます。このIDは次のように使います:

```
GET /rest/Employee/$entityset/9718A30BF61343C796345F3BE5B01CE7
```

\$entityset/{entitySetID}?\$operator...&\$otherCollection

複数の既存エンティティセットに基づいて新たなエンティティセットを作成します。

引数	タイプ	説明
<code>\$operator</code>	String	既存のエンティティセットに対して使用する論理演算子
<code>\$otherCollection</code>	String	エンティティセットID

説明

`$method=entityset` を使ってエンティティセット(エンティティセット#1)を作成したあとで、`$entityset/{entitySetID}?$operator...&$otherCollection` シンタックスを使って新たなエンティティセットを作成できます。このとき、`$operator` に指定できる値は後述のとおりで、2つ目のエンティティセット(エンティティセット#2)は `$otherCollection` プロパティに指定します。2つのエンティティセットは同じデータクラスに属していないかもしれません。

このリクエストの結果を格納するエンティティセットを作成する場合は、RESTリクエストの最後に `$method=entityset` を追加します。

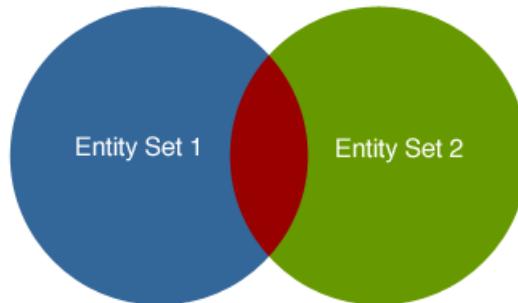
下記は、論理演算子の一覧です:

演算子	説明
AND	両方のエンティティセットに共通して含まれるエンティティのみを返します。
OR	両エンティティセットのいずれか、あるいは両方に含まれているエンティティを返します。
EXCEPT	エンティティセット#1 から、エンティティセット#2にも含まれているエンティティを除外した残りを返します。
INTERSECT	両方のエンティティセットに共通して含まれるエンティティがあれば true、なければ false を返します。

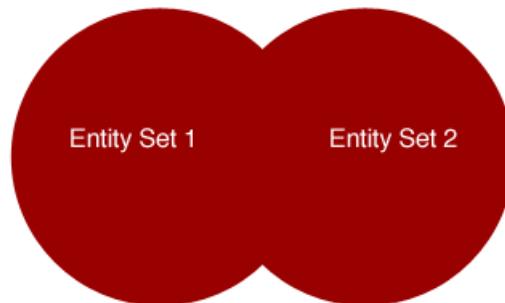
論理演算子の文字の大小は区別されないため、"AND" とも "and" とも書けます。

2つのエンティティセットを対象に論理演算子を使用した場合のベン図は下のとおりです。赤く塗られた部分が返されるものです。

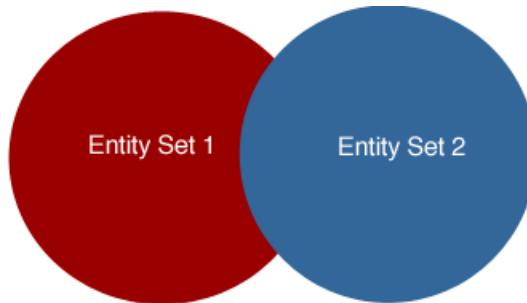
AND



OR



EXCEPT



シンタクスは次のとおりです:

```
GET /rest/dataClass/$entityset/entitySetID?$logicOperator=AND&$otherCollection=entitySetID
```

例題

次の例では AND論理演算子を使用するため、両方のエンティティセットに共通して含まれるエンティティが返されます:

```
GET /rest/Employee/$entityset/9718A30BF61343C796345F3BE5B01CE7?
$logicOperator=AND&$otherCollection=C05A0D887C664D4DA1B38366DD21629B
```

2つのエンティティセットが交差するかどうかを確認するには、次のように書きます:

```
GET /rest/Employee/$entityset/9718A30BF61343C796345F3BE5B01CE7?
$logicOperator=intersect&$otherCollection=C05A0D887C664D4DA1B38366DD21629B
```

共通のエンティティが存在する場合、このクエリは true を返します。それ以外の場合は false を返します。

次の例では、2つのエンティティセットのいずれかあるいは両方に含まれているエンティティすべてを格納した新しいエンティティセットを作成します：

```
GET /rest/Employee/$entityset/9718A30BF61343C796345F3BE5B01CE7?  
$logicOperator=OR&$otherCollection=C05A0D887C664D4DA1B38366DD21629B&$method=entityset
```

\$expand

画像属性に保存されているピクチャーを展開します (例: `Employee(1)/photo?$imageformat=best&$expand=photo`)

または

保存するために BLOB属性を展開します。

互換性に関する注記: 互換性のため、\$expand はリレーション属性を展開するのに使用できます (例: `Company(1)?$expand=staff` または `Employee/?$filter="firstName BEGIN a"&$expand=employer`)。しかしながら、これらの場合には `$attributes` を使用するのが推奨されます。

画像属性の表示

画像属性の全体像を表示させるには、次のように書きます:

```
GET /rest/Employee(1)/photo?$imageformat=best&$version=1&$expand=photo
```

画像形式についての詳細は `$imageformat` を参照ください。version パラメーターについての詳細は `$version` を参照ください。

BLOB属性のディスク保存

データクラスに保存されている BLOB をディスクに保存するには、\$binary に "true" を渡すことで、次のように書けます:

```
GET /rest/Company(11)/blobAtt?$binary=true&$expand=blobAtt
```

\$filter

データクラスまたはメソッドが返すデータをフィルターします (例: `$filter="firstName!='' AND salary>30000"`)

説明

このパラメーターを使って、データクラスまたはメソッドが返すデータに対するフィルターを定義することができます。

単純なフィルターの利用

フィルターは次の要素で構成されます:

```
{attribute} {comparator} {value}
```

たとえば `$filter="firstName=john"` の場合、`firstName` は属性 (attribute)、`=` は比較演算子 (comparator)、`john` は値 (value) にあたります。

複雑なフィルターの利用

複雑なフィルターは複数の単純なフィルターの組み合わせで構成されます:

```
{attribute} {comparator} {value} {AND/OR/EXCEPT} {attribute} {comparator} {value}
```

たとえば: `$filter="firstName=john AND salary>20000"` (`firstName` および `salary` は Employee データクラスの属性です)。

paramsプロパティの使用

4D の params プロパティを使うこともできます。

```
{attribute} {comparator} {placeholder} {AND/OR/EXCEPT} {attribute} {comparator} {placeholder}&$params='["{value1}","{value2}"]'
```

たとえば: `$filter="firstName=:1 AND salary:>:2" &$params='["john",20000]'` (`firstName` および `salary` は Employee データクラスの属性です)。

4D においてデータをクエリする方法についての詳細は、[dataClass.query\(\)](#) ドキュメンテーションを参照ください。

单一引用符 ('') または二重引用符 ("") を挿入するには、対応する文字コードを使ってそれらをエスケープする必要があります:

- 単一引用符 (''): \u0027
- 二重引用符 (""): \u0022

たとえば、单一引用符が含まれる値を `params` プロパティに渡すには、次のように書きます:

```
http://127.0.0.1:8081/rest/Person/?$filter="lastName=:1" &$params='["0\u0027Reilly"]'
```

値を直接渡す場合は、次のように書けます: `http://127.0.0.1:8081/rest/Person/?$filter="lastName=0'Reilly"`

属性

同じデータクラスに属している属性はそのまま受け渡せます (例: `firstName`)。別のデータクラスをクエリする場合は、リレーション名と属性、つまりパスを渡さなくてはなりません (例: `employer.name`)。属性名の文字の大小は区別されます (`firstName` と `FirstName` は異なります)。

オブジェクト型属性もドット記法によってクエリできます。たとえば、"objAttribute" という名称のオブジェクト属性が次の構造を持っていた場合:

```
{  
    prop1: "一つ目のプロパティです",  
    prop2: 9181,  
    prop3: ["abc", "def", "ghi"]  
}
```

このオブジェクトをクエリするには、次のように書きます:

```
GET /rest/Person/?filter="objAttribute.prop2 == 9181"
```

比較演算子

以下の比較演算子を使用できます:

比較演算子	説明
=	等しい
!=	等しくない
>	大きい
>=	以上
<	小さい
<=	以下
begin	前方一致

例題

名字が "j" で始まる社員を検索します:

```
GET /rest/Employee?$filter="lastName begin j"
```

Employee データクラスより、給与が 20,000 超で、かつ Acme という名称の企業で働いていない社員を検索します:

```
GET /rest/Employee?$filter="salary>20000 AND  
employer.name!=acme"&$orderby="lastName,firstName"
```

Person データクラスより、anotherobj オブジェクト属性の number プロパティが 50 より大きい人のデータを検索します:

```
GET /rest/Person/?filter="anotherobj.mynum > 50"
```

\$imageformat

画像取得の際に使用する画像形式を指定します (例: `$imageformat=png`)

説明

画像の表示に使う形式を指定します。デフォルトでは、画像に最適な形式が選択されます。指定する場合は、次の形式が指定できます:

タイプ	説明
GIF	GIF 形式
PNG	PNG 形式
JPEG	JPEG 形式
TIFF	TIFF 形式
best	画像に最適な形式

画像を完全に読み込むには、形式を指定するだけでなく、画像属性を `$expand` に渡す必要があります。

読み込むべき画像がない場合、または指定した形式では画像が読み込めない場合、レスポンスは空になります。

例題

photo属性の実際の形式に関わらず、画像形式を JPEG に指定し、サーバーより受け取ったバージョン番号を受け渡している例です:

```
GET /rest/Employee(1)/photo?$imageformat=jpeg&$version=3&$expand=photo
```

\$lock

[ペシミスティック・ロック機構](#) を使ってエンティティをロック/アンロックします。

シンタックス

他のセッションや 4Dプロセスに対し、特定のエンティティをロックするには:

```
/?$lock=true
```

他のセッションや 4Dプロセスに対し、特定のエンティティをアンロックするには:

```
/?$lock=false
```

`lockKindText` プロパティ は "Locked by session" です。

説明

REST API によるロックは、[セッション](#) レベルで設定されます。

ロックされたエンティティは次のものから操作（ロック / アンロック / 更新 / 削除）できません:

- 他の REST セッション
- RESTサーバー上で実行されている 4D プロセス（クライアント/サーバー, リモートデータストア, スタンドアロン）。

REST API によってロックされたエンティティは、次の場合にのみアンロックされます:

- ロック元の（`/?$lock=true` をおこなった）RESTセッションが `/?$lock=false` をリクエストしたとき。
- ロック元セッションの [非アクティブタイムアウト](#) に達したとき（セッションは閉じられます）。

レスポンス

`?$lock` リクエストでロック操作に成功した場合、`"result":true` を格納した JSONオブジェクトが返されます（失敗した場合は `"result":false`）。

戻り値の `"__STATUS"` オブジェクトには、以下のプロパティが格納されています:

プロパティ		タイプ	説明
			成功の場合にのみ利用可能:
success		boolean	ロックに成功した場合 (あるいはエンティティがすでにカレントセッションでロックされていた場合) には true、それ以外は false (この場合は返されません)。
			エラーの場合にのみ利用可能:
status		number	エラーコード、以下参照
statusText		テキスト	エラーの詳細、以下参照
lockKind		number	ロックコード
lockKindText		テキスト	"Locked by session" RESTセッションによるロック、"Locked by record" 4Dプロセスによるロック
lockInfo		object	ロック元についての情報。返されるプロパティはロック元 (4Dプロセスまたは RESTセッション) によって異なります。
			4Dプロセスによるロックの場合:
	task_id	number	プロセスID
	user_name	テキスト	マシン上でのセッションユーザー名
	user4d_alias	テキスト	4D ユーザーの名前またはエイリアス
	user4d_id	number	4DデータベースディレクトリでのユーザーID
	host_name	テキスト	マシン名
	task_name	テキスト	プロセス名
	client_version	テキスト	クライアントのバージョン
			RESTセッションによるロックの場合:
	host	テキスト	エンティティをロックした URL (例: "127.0.0.1:8043")
	IPAddr	テキスト	ロック元の IPアドレス (例: "127.0.0.1")
	recordNumber	number	ロックされたレコードのレコード番号
	userAgent	テキスト	ロック元の userAgent (例: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36")

エラー時には __STATUS オブジェクトの *status* および *statusText* プロパティに以下のいずれかの値が返されます:

status	statusText	説明
2	"Stamp has changed"	エンティティの内部的なスタンプ値がデータ内に保存されているエンティティのものと合致しません (オプティミスティック・ロック)。
3	"Already locked"	エンティティはペシミスティック・ロックでロックされています。
4	"Other error"	深刻なエラーとは、低レベルのデータベースエラー (例: 重複キー)、ハードウェアエラーなどです。
5	"Entity does not exist anymore"	エンティティはもうデータ内に存在していません。

例題

一つ目のブラウザからエンティティをロックします:

```
GET /rest/Customers(1)/?$lock=true
```

レスポンス:

```
{  
    "result": true,  
    "__STATUS": {  
        "success": true  
    }  
}
```

二つ目のブラウザ（別のセッション）から、同じリクエストを送信します：

レスポンス:

```
{  
    "result":false,  
    "__STATUS":{  
        "status":3,  
        "statusText":"Already Locked",  
        "lockKind":7,  
        "lockKindText":"Locked By Session",  
        "lockInfo":{  
            "host":"127.0.0.1:8043",  
            "IPAddr":"127.0.0.1",  
            "recordNumber": 7,  
            "userAgent": """Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_3) AppleWebKit/537.36..."  
        }  
    }  
}
```

\$method

このパラメーターは、返されたエンティティまたはエンティティセレクションに対して実行する処理を指定するのに使います。

使用可能なシンタックス

シンタックス	例題	説明
\$method=delete	POST /Employee?\$filter="ID=11" & \$method=delete	エンティティまたはエンティティセレクションを削除します
\$method=entityset	GET /People/?\$filter="ID>320" & \$method=entityset & \$timeout=600	RESTリクエストで定義されたエンティティのコレクションに基づいて、4D Server のキャッシュにエンティティセットを作成します
\$method=release	GET /Employee/\$entityset/<entitySetID>? \$method=release	4D Server のキャッシュからエンティティセットを削除します
\$method=subentityset	GET /Company(1)/staff?\$expand=staff& \$method=subentityset& \$subOrderby=lastName ASC	RESTリクエストで定義されたリレートエンティティのコレクションに基づいて、エンティティセットを作成します
\$method=update	POST /Person/?\$method=update	一つ以上のエンティティを更新または作成します

\$method=delete

エンティティまたは (RESTで作成された) エンティティセレクションを削除します

説明

\$method=delete を使ってエンティティ、またはエンティティセレクションを削除します。たとえば、\$filter を使って定義したエンティティセレクションや、{dataClass}({key}) (例: /Employee(22)) のように直接特定したエンティティが対象です。

\$entityset/{entitySetID} のようにエンティティセットを呼び出して、そこに含まれるエンティティを削除することもできます。

例題

キーが 22 であるエンティティを削除するには、次の RESTリクエストが書けます:

```
POST /rest/Employee(22)/?$method=delete
```

\$filter を使ったクエリも可能です:

```
POST /rest/Employee?$filter="ID=11" & $method=delete
```

\$entityset/{entitySetID} で呼び出したエンティティセットを削除する場合は次のように書きます:

```
POST /rest/Employee/$entityset/73F46BE3A0734EAA9A33CA8B14433570? $method=delete
```

レスポンス:

```
{
    "ok": true
}
```

\$method=entityset

RESTリクエストで定義されたエンティティのコレクションに基づいて、4D Server のキャッシュにエンティティセットを作成します

説明

RESTでエンティティのコレクションを作成した場合、これをエンティティセットとして 4D Server のキャッシュに保存することができます。エンティティセットには参照番号が付与されます。これを `$entityset/{entitySetID}` に渡すと、当該エンティティセットにアクセスできます。デフォルトで、エンティティセットは 2時間有効です。`$timeout` に値(秒単位)を渡すことで、有効時間を変更できます。

エンティティセットを作成する際に、`$filter` や `$orderby` と同時に `$savedfilter` や `$savedorderby` も使用していた場合には、4D Server のキャッシュからエンティティセットが削除されていても、同じ参照IDで再作成できます。

例題

4D Server のキャッシュに2時間保存されるエンティティセットを作成するには、RESTリクエストの最後に `$method=entityset` を追加します:

```
GET /rest/People/?$filter="ID>320"&$method=entityset
```

保存時間が 10分のエンティティセットを作成するには、次のように `$timeout` に値を渡します:

```
GET /rest/People/?$filter="ID>320"&$method=entityset&$timeout=600
```

フィルターや並べ替えの情報を保存するには、`$savedfilter` や `$savedorderby` に true を渡します。

エンティティセットを作成する際には、`$skip` および `$top/$limit` は無視されます。

エンティティセットを作成すると、返されるオブジェクトの先頭に `__ENTITYSET` という要素が追加され、エンティティセットにアクセスするための URI を提供します:

```
__ENTITYSET: "http://127.0.0.1:8081/rest/Employee/$entityset/9718A30BF61343C796345F3BE5B01CE7"
```

\$method=release

4D Server のキャッシュからエンティティセットを削除します。

説明

`$method=entityset` によって作成したエンティティセットを、4D Server のキャッシュから削除することができます。

例題

既存のエンティティセットを削除します:

```
GET /rest/Employee/$entityset/4C51204DD8184B65AC7D79F09A077F24?$method=release
```

レスポンス:

リクエストが成功した場合のレスポンス:

```
{  
    "ok": true  
}  
エンティティセットが見つからなかった場合には、エラーが返されます  
  
{  
    "__ERROR": [  
        {  
            "message": "Error code: 1802\\nEntitySet \\\"4C51204DD8184B65AC7D79F09A077F24\\\" cannot be found",  
            "componentSignature": "dbmg",  
            "errCode": 1802  
        }  
    ]  
}
```

\$method=subentityset

RESTリクエストで定義されたリレートエンティティのコレクションに基づいて、4D Server のキャッシュにエンティティセットを作成します

説明

`$method=subentityset` を使うことで、RESTリクエストが定義されたリレーション属性によって返されるデータを並べ替えることができます。

データを並べ替えるには `$suborderby` を使います。並べ替えの基準とする各属性について、並べ替え順を指定します。ASC (`asc`) が昇順、DESC (`desc`) が降順です。デフォルトでは、データは昇順に並べ替えられます。

複数の属性を指定するには、カンマ区切りにします (例 : `$suborderby="lastName desc, firstName asc"`)。

例題

あるエンティティのリレートエンティティだけを取得したいとき、たとえば Company データクラスの staff リレーション名が Employee データクラスにリンクしている場合には、次の RESTリクエストが書けます:

```
GET /rest/Company(1)/staff?$expand=staff&$method=subentityset&$suborderby=lastName ASC
```

レスポンス:

```
{
    "__ENTITYSET": "/rest/Employee/$entityset/FF625844008E430B9862E5FD41C741AB",
    "__entityModel": "Employee",
    "__COUNT": 2,
    "__SENT": 2,
    "__FIRST": 0,
    "__ENTITIES": [
        {
            "__KEY": "4",
            "__STAMP": 1,
            "ID": 4,
            "firstName": "Linda",
            "lastName": "Jones",
            "birthday": "1970-10-05T14:23:00Z",
            "employer": {
                "__deferred": {
                    "uri": "/rest/Company(1)",
                    "__KEY": "1"
                }
            }
        },
        {
            "__KEY": "1",
            "__STAMP": 3,
            "ID": 1,
            "firstName": "John",
            "lastName": "Smith",
            "birthday": "1985-11-01T15:23:00Z",
            "employer": {
                "__deferred": {
                    "uri": "/rest/Company(1)",
                    "__KEY": "1"
                }
            }
        }
    ]
}
```

\$method=update

一つ以上のエンティティを更新または作成します

説明

`$method=update` を使うと、一つの POST で一つ以上のエンティティを更新または作成することができます。エンティティの更新・作成をおこなうには、オブジェクトのプロパティ/値としてエンティティの属性/値を指定します（例：`{ lastName: "Smith" }`）。複数のエンティティを更新・作成するには、各エンティティに対応するオブジェクトをコレクションにまとめます。

いずれの場合も、リクエストのボディ（body）に POST データ body を格納します。

エンティティを更新するには、更新する属性だけでなく、`__KEY` および `__STAMP` パラメーターをオブジェクト内に指定しなくてはなりません。これらのパラメーターがない場合、POST のボディに格納したオブジェクトの値をもとに新規エンティティが追加されます。

エンティティをサーバーに保存すると同時にトリガーが実行されます。レスポンスにはすべてのデータが、サーバー上に存在するとおりに格納されます。

`$atomic/$atonce` を使うと、エンティティを作成・更新するリクエストをトランザクション内で実行できます。データの検証でエラーが発生した場合に、一部のエンティティだけが処理されてしまうのを防げます。また、`$method=validate` を使うと、作成・更新の前にエンティティを検証することができます。

エンティティを追加または更新する際に問題が発生すると、その情報を格納したエラーが返されます。

属性の型に関する注記:

- 日付は JavaScript 形式で表す必要があります: YYYY-MM-DDTHH:MM:SSZ (例: "2010-10-05T23:00:00Z")。日付属性のためだけに日付プロパティを指定した場合、タイムゾーンおよび時刻 (時間・分・秒) の情報は削除されます。この場合、レスポンスの形式 dd!mm!yyyy (例: 05!10!2013) を使って日付を送信することも可能です。
- 布尔は true または false です。
- `$upload` を使ってアップロードしたファイルは、{ "ID": "D507BC03E613487E9B4C2F6A0512FE50" } のような形式で返されるオブジェクトを渡すことで、ピクチャー型やBLOB型の属性に適用できます。

例題

特定のエンティティを更新するには、次のようなリクエストをします:

```
POST /rest/Person/?$method=update
```

POST データ:

```
{
  __KEY: "340",
  __STAMP: 2,
  firstName: "Pete",
  lastName: "Miller"
}
```

この場合、渡した firstName および lastName 属性だけが変更され、当該エンティティのその他の属性はそのままです (変更した属性に基づいて計算される属性を除く)。

エンティティを作成するには、次のように書きます:

```
POST /rest/Person/?$method=update
```

POST データ:

```
{
  firstName: "John",
  lastName: "Smith"
}
```

同じ URL を使って、複数のエンティティを作成・更新することもできます。その場合には、POST データに複数オブジェクトのコレクションを渡します:

```
POST /rest/Person/?$method=update
```

POST データ:

```
[{
  "__KEY": "309",
  "__STAMP": 5,
  "ID": "309",
  "firstName": "Penelope",
  "lastName": "Miller"
}, {
  "firstName": "Ann",
  "lastName": "Jones"
}]
```

レスポンス:

エンティティを追加・更新した場合、そのエンティティは変更後の内容で返されます。たとえば、新規の Employee エンティティを作成した場合、次のようなレスポンスが返されます:

```
{  
    "_KEY": "622",  
    "_STAMP": 1,  
    "uri": "http://127.0.0.1:8081/rest/Employee(622)",  
    "_TIMESTAMP": "!!2020-04-03!!",  
    "ID": 622,  
    "firstName": "John",  
    "firstName": "Smith"  
}
```

スタンプが正しくない場合には、次のようなエラーが返されます：

```
{
    "__STATUS": {
        "status": 2,
        "statusText": "Stamp has changed",
        "success": false
    },
    "__KEY": "1",
    "__STAMP": 12,
    "__TIMESTAMP": "!!2020-03-31!!",
    "ID": 1,
    "firstname": "Denise",
    "lastname": "O'Peters",
    "isWoman": true,
    "numberOfKids": 1,
    "addressID": 1,
    "gender": true,
    "imageAtt": {
        "__deferred": {
            "uri": "/rest/Persons(1)/imageAtt?$imageformat=best&$version=12&$expand=imageAtt",
            "image": true
        }
    },
    "extra": {
        "num": 1,
        "alpha": "I am 1"
    },
    "address": {
        "__deferred": {
            "uri": "/rest/Address(1)",
            "__KEY": "1"
        }
    },
    "__ERROR": [
        {
            "message": "Given stamp does not match current one for record# 0 of table Persons",
            "componentSignature": "dbmg",
            "errCode": 1263
        },
        {
            "message": "Cannot save record 0 in table Persons of database remote_dataStore",
            "componentSignature": "dbmg",
            "errCode": 1046
        },
        {
            "message": "The entity# 1 in the \"Persons\" dataclass cannot be saved",
            "componentSignature": "dbmg",
            "errCode": 1517
        }
    ]
}{}}
}
```

\$orderby

指定した属性と並べ替え順に基づいて、返されたデータを並べ替えます（例: `$orderby="lastName desc, salary asc"`）

説明

`$orderby` は RESTリクエストによって返されるエンティティを並べ替えます。並べ替えの基準とする各属性について、並べ替え順を指定します。`ASC` (`asc`) が昇順、`DESC` (`desc`) が降順です。デフォルトでは、データは昇順に並べ替えられます。複数の属性を指定するには、カンマ区切りにします 例: `$orderby="lastName desc, firstName asc"`。

例題

取得と同時にエンティティを並べ替えます:

```
GET /rest/Employee/?$filter="salary!=0"&$orderby="salary DESC,lastName ASC,firstName ASC"
```

以下の例では、`lastName`属性を基準にしてエンティティセットを昇順に並べ替えます:

```
GET /rest/Employee/$entityset/CB1BCC603DB0416D939B4ED379277F02?$orderby="lastName"
```

結果:

```
{
  __entityModel: "Employee",
  __COUNT: 10,
  __SENT: 10,
  __FIRST: 0,
  __ENTITIES: [
    {
      __KEY: "1",
      __STAMP: 1,
      firstName: "John",
      lastName: "Smith",
      salary: 90000
    },
    {
      __KEY: "2",
      __STAMP: 2,
      firstName: "Susan",
      lastName: "O'Leary",
      salary: 80000
    },
    ...
  ]
}
```

\$querypath

4D Server によって実際に実行されたクエリを返します（例: `$querypath=true`）

説明

`$querypath` は、4D Server によって実際に実行されたクエリを返します。たとえば、クエリの一部がエンティティを返さなかった場合、残りのクエリは実行されません。`$querypath` で確認されるとおり、クエリリクエストは最適化されます。

クエリパスについての詳細は [queryPlan](#) と [queryPath](#) を参照ください。

実行されたクエリを定義する次のプロパティを格納した `steps` コレクションが返されます：

プロパティ	タイプ	説明
<code>description</code>	<code>String</code>	実際に実行されたクエリ、または複数ステップの場合は "AND"
<code>time</code>	数値	クエリの実行に要した時間 (ミリ秒単位)
<code>recordsfounds</code>	数値	レコードの検出件数
<code>steps</code>	コレクション	クエリパスの後続ステップを定義するオブジェクトのコレクション

例題

以下のクエリを渡した場合：

```
GET /rest/Employee/$filter="employer.name=acme AND lastName=Jones"&$querypath=true
```

エンティティが見つからなかった場合に次のように書くと、後述のクエリパスが返されます：

```
GET /rest/$querypath
```

レスポンス：

```

__queryPath: {

  steps: [
    {
      description: "AND",
      time: 0,
      recordsfounds: 0,
      steps: [
        {
          description: "Join on Table : Company : People.employer = Company.ID",
          time: 0,
          recordsfounds: 0,
          steps: [
            {
              steps: [
                {
                  description: "Company.name = acme",
                  time: 0,
                  recordsfounds: 0
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}

```

最初のクエリが一つ以上のエンティティを返した場合には、二つめのクエリが実行されます。以下のクエリを実行した場合:

```
GET /rest/Employee/$filter="employer.name=a* AND lastName!=smith"&$querypath=true
```

少なくとも1件のエンティティが見つかった場合に次のように書くと、後述のクエリパスが返されます:

```
GET /rest/$querypath
```

レスポンス:

```
"__queryPath": {
  "steps": [
    {
      "description": "AND",
      "time": 1,
      "recordsfounds": 4,
      "steps": [
        {
          "description": "Join on Table : Company : Employee.employer = Company.ID",
          "time": 1,
          "recordsfounds": 4,
          "steps": [
            {
              "steps": [
                {
                  "description": "Company.name LIKE a*",
                  "time": 0,
                  "recordsfounds": 2
                }
              ]
            }
          ]
        },
        {
          "description": "Employee.lastName # smith",
          "time": 0,
          "recordsfounds": 4
        }
      ]
    }
  ]
}
```

\$queryplan

4D Server に渡したクエリを返します (例: `$queryplan=true`)

説明

\$queryplan は、4D Server に渡したクエリプランを返します。

プロパティ	タイプ	説明
item	String	渡された実際のクエリ
subquery	配列	(サブクエリが存在する場合) item プロパティを格納する追加のオブジェクト

クエリプランについての詳細は [queryPlan](#) と [queryPath](#) を参照ください。

例題

以下のクエリを渡した場合:

```
GET /rest/People/$filter="employer.name=acme AND lastName=Jones"&$queryplan=true
```

レスポンス:

```
__queryPlan: {
    And: [
        {
            item: "Join on Table : Company : People.employer = Company.ID",
            subquery: [
                {
                    item: "Company.name = acme"
                }
            ]
        },
        {
            item: "People.lastName = Jones"
        }
    ]
}
```

\$savedfilter

エンティティセット作成時に、\$filter に定義したフィルターを保存します（例： \$savedfilter="{filter}" ）

説明

エンティティセットを作成する際に使用したフィルターを念のために保存しておくことができます。4D Server のキャッシュからエンティティセットが削除されてしまって（たとえばタイムアウトや容量の問題、 \$method=release 操作によって）、同じエンティティセットを取り戻すことができます。

\$savedfilter を使用してエンティティセット作成時に使ったフィルターを保存したあとは、エンティティセットを取得する度に \$savedfilter も受け渡します。

4D Server のキャッシュからエンティティセットが消えていた場合、10分のデフォルトタイムアウトで再作成されます。エンティティセットが消えていた場合、再作成されるエンティティセットの内容は更新されたものです（新しくエンティティが追加されていたり、存在していたエンティティが削除されていたりする場合があります）。

エンティティセットの作成時に \$savedfilter と \$savedorderby の両方を使用したにも関わらず、次の呼び出しでは片方を省略すると、返されるエンティティセットは同じ参照番号を持ちながら、この変更を反映します。

例題

エンティティセットを作成する際に \$savedfilter を使います：

```
GET /rest/People/?$filter="employer.name=Apple"&$savedfilter="employer.name=Apple"&$method=entityset
```

作成したエンティティセットにアクセスする際、そのエンティティセットが有効なのを確実にしたい場合には、次のように書きます：

```
GET /rest/People/$entityset/AEA452C2668B4F6E98B6FD2A1ED4A5A8?$savedfilter="employer.name=Apple"
```

\$savedorderby

エンティティセット作成時に、`$orderby` に定義した並べ替え情報を保存します（例： `$savedorderby="{orderby}"` ）

説明

エンティティセットを作成する際に使用した並べ替え情報を念のために保存しておくことができます。4D Server のキャッシュからエンティティセットが削除されてしまっても（たとえばタイムアウトや容量の問題、`$method=release` 操作によって）、同じエンティティセットを取り戻すことができます。

`$savedorderby` を使用してエンティティセット作成時に使った並べ替え情報を保存したあとは、エンティティセットを取得する度に `$savedorderby` も受け渡します。

4D Server のキャッシュからエンティティセットが消えていた場合、10分のデフォルトタイムアウトで再作成されます。エンティティセットの作成時に `$savedfilter` と `$savedorderby` の両方を使用したにも関わらず、次の呼び出しでは片方を省略すると、返されるエンティティセットは同じ参照番号を持ちながら、この変更を反映します。

例題

エンティティセットを作成する際に `$savedorderby` を使います：

```
GET /rest/People/?  
$filter="lastName!=""&$savedfilter="lastName!=""&$orderby="salary"&$savedorderby="salary"&$method=entity  
set
```

作成したエンティティセットにアクセスする際、そのエンティティセットが有効なのを確実にしたい場合には、（`$savedfilter` と `$savedorderby` を両方使って）次のように書きます：

```
GET /rest/People/$entityset/AEA452C2668B4F6E98B6FD2A1ED4A5A8?  
$savedfilter="lastName!=""&$savedorderby="salary"
```

\$skip

エンティティセレクション内で、この数値によって指定されたエンティティから処理を開始します（例: `$skip=10`）

説明

`$skip` はセレクション内のどのエンティティから処理を開始するかを指定します。デフォルトでは、先頭エンティティから開始します。10番目のエンティティから開始するには、10を渡します。

`$skip` は通常、`$top/$limit` との組み合わせで使用され、エンティティセレクション内をナビゲートするのに使います。

例題

エンティティセットの20番目のエンティティ以降を取得します：

```
GET /rest/Employee/$entityset/CB1BCC603DB0416D939B4ED379277F02?$skip=20
```

\$timeout

4D Server のキャッシュにエンティティセットが保存される時間 (秒単位) を指定します (例: `$timeout=1800`)

説明

`$method=entityset` を使って作成するエンティティセットについてタイムアウトを指定するには任意の秒数を `$timeout` に渡します。たとえば、20分のタイムアウトを設定するには、1200を渡します。デフォルトのタイムアウトは 2時間です。

一旦タイムアウトが定義されると、(`$method=entityset` によって) エンティティセットが呼び出される度に現時刻とタイムアウトに基づいて有効期限が再計算されます。

一度削除されたエンティティセットが `$method=entityset` と `$savedfilter` の組み合わせで再作成された場合、`$timeout` で指定している数値に関わらず、デフォルトタイムアウトは 10分です。

例題

作成するエンティティセットのタイムアウトを 20分に設定します:

```
GET /rest/Employee/?$filter="salary!=0"&$method=entityset&$timeout=1200
```

\$top/\$limit

返されるエンティティの数を制限します (例: `$top=50`)

説明

`$top/$limit` は返されるエンティティの数を制限します。この数字はデフォルトで 100 件です。 `$top` および `$limit` のどちらでも利用できます。

`$skip` と組み合わせて使用すると、RESTリクエストによって返されるエンティティセレクション内を移動することができます。

例題

エンティティセットから、20 番目以降の 10 件のエンティティを取得します:

```
GET /rest/Employee/$entityset/CB1BCC603DB0416D939B4ED379277F02?$skip=20&$top=10
```

\$version

画像のバージョン番号

説明

\$version はサーバーにより返される画像のバージョン番号です。サーバーより受け取るバージョン番号は、ブラウザーのキャッシュを回避し、正しい画像を取得できるようにします。

画像の version パラメーターの値はサーバーによって変更されます。

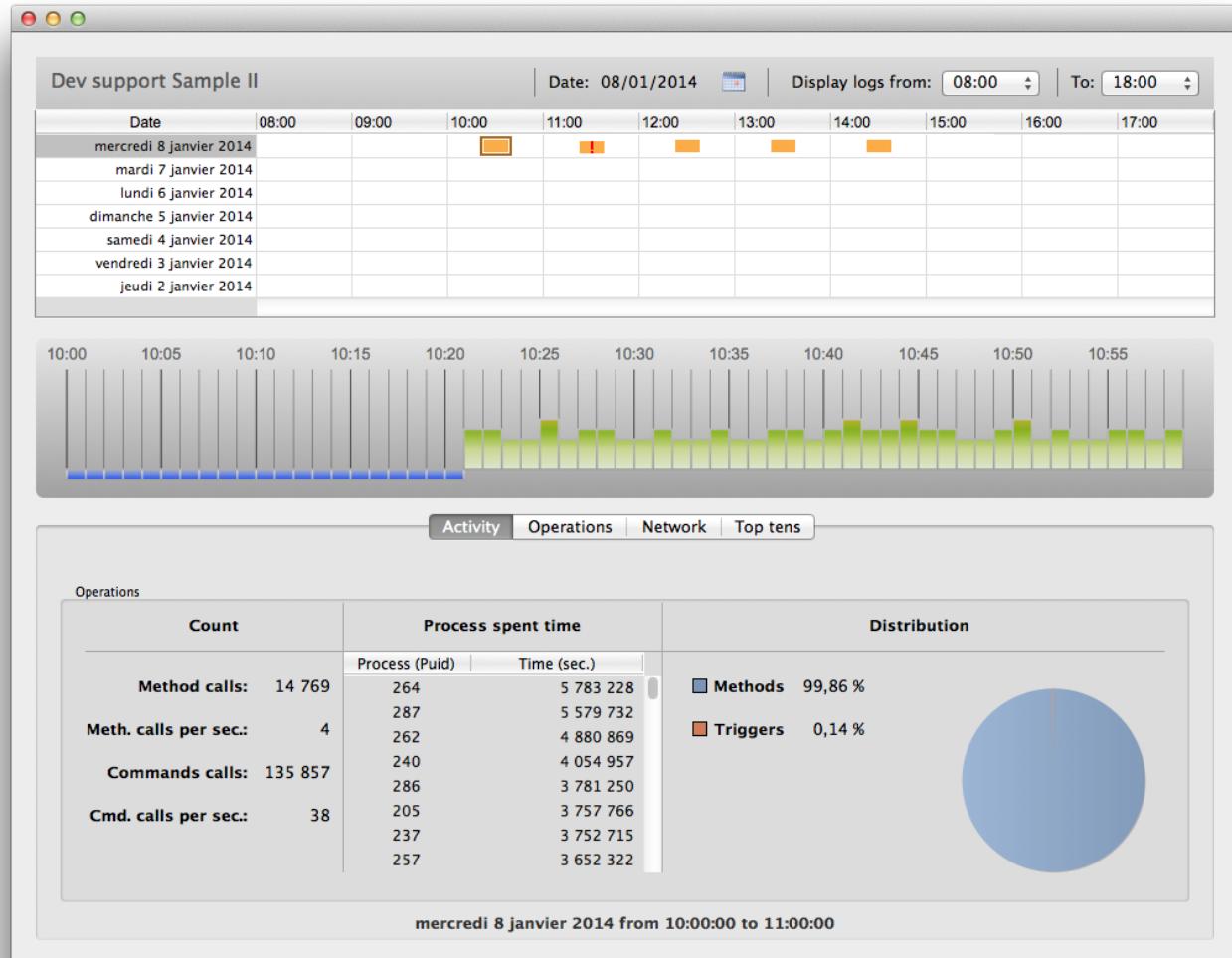
例題

photo属性の実際の形式に関わらず、画像形式を JPEG に指定し、サーバーより受け取ったバージョン番号を受け渡している例です：

```
GET /rest/Employee(1)/photo?$imageformat=jpeg&$version=3&$expand=photo
```

4D フォームについて

フォームはデスクトップアプリケーションにおいて、データの入力・修正・印刷をおこなうためのインターフェースとなります。フォームを使用することで、ユーザーはデータベースのデータを取り出し、レポートを印刷します。フォームを使用して、カスタムダイアログボックスやパレット、そのほかのカスタムウィンドウを作成します。



また、以下の機能により、フォームは他のフォームを含むことができます：

- サブフォームオブジェクト
- 継承フォーム

フォームを作成する

4Dフォームの追加や変更は、以下の要素を使っておこないます：

- 4D Developer インターフェース：ファイル メニューまたは エクスプローラ ウィンドウから新規フォームを作成できます。
- フォームエディター：フォームの編集は [フォームエディター](#) を使っておこないます。
- JSON コード：JSON を使ってフォームを作成・設計し、フォーム ファイルを [適切な場所](#) に保存します。例：

```
{
    "windowTitle": "Hello World",
    "windowMinWidth": 220,
    "windowMinHeight": 80,
    "method": "HWexample",
    "pages": [
        null,
        {
            "objects": {
                "text": {
                    "type": "text",
                    "text": "Hello World!",
                    "textAlign": "center",
                    "left": 50,
                    "top": 120,
                    "width": 120,
                    "height": 80
                },
                "image": {
                    "type": "picture",
                    "pictureFormat": "scaled",
                    "picture": "/RESOURCES/Images/HW.png",
                    "alignment": "center",
                    "left": 70,
                    "top": 20,
                    "width": 75,
                    "height": 75
                },
                "button": {
                    "type": "button",
                    "text": "OK",
                    "action": "Cancel",
                    "left": 60,
                    "top": 160,
                    "width": 100,
                    "height": 20
                }
            }
        }
    ]
}
```

プロジェクトフォームとテーブルフォーム

2つのカテゴリーのフォームが存在します:

- プロジェクトフォーム - テーブルに属さない独立したフォームです。このタイプのフォームは、おもにインターフェースダイアログボックスやコンポーネントを作成するのに使用されます。プロジェクトフォームを使用してより簡単に OS 標準に準拠するインターフェースを作成できます。
- テーブルフォーム - 特定のテーブルに属していて、それによりデータベースに基づくアプリケーションの開発に便利な自動機能の恩恵を得ることができます。通常、テーブルには入力フォームと出力フォームが別々に存在します。

フォームを作成する際にフォームカテゴリーを選択しますが、後から変更することも可能です。

フォームのページ

各フォームは、少なくとも 2つのページで構成されています:

- ページ1: デフォルトで表示されるメインページ
- ページ0: 背景ページ。このページ上に置かれたオブジェクトはすべてのページで表示されます

1つの入力フォームに複数のページを作成することができます。一画面に納まりきらない数のフィールドや変数がある場合は、これらを表示するためにページを追加することができます。複数のページを作成すると、以下のようなことが可能になります:

- もっとも重要な情報を最初のページに配置し、他の情報を後ろのページに配置する。
- トピックごとに、専用ページにまとめる。
- [入力順](#)を設定して、データ入力中のスクロール動作を少なくしたり、または不要にする。
- フォーム要素の周りの空間を広げ、洗練された画面をデザインする。

複数ページは入力フォームとして使用する場合にのみ役立ちます。印刷出力には向きません。マルチページフォームを印刷すると、最初のページしか印刷されません。

フォームのページ数には制限がありません。フォーム内の複数ページ上に同じフィールドを何度も表示することができます。しかし、フォームのページ数が多くなるほど、フォームの表示に要する時間が長くなります。

マルチページフォームには、1つの背景ページと複数の表示ページが存在します。背景ページ上に置かれたオブジェクトはすべての表示ページに現れます。それらのオブジェクトの選択や編集は背景ページでのみ可能です。複数ページフォームでは、ボタンパレットを背景ページに置くべきです。また、ページ移動ツールオブジェクトを背景ページに配置し、ユーザーに提供する必要があります。

継承フォーム

4D では "継承フォーム" を使用することができます。これはつまり、フォームA の全オブジェクトが フォームB で使用可能であるということです。この場合、フォームB は フォームA からオブジェクトを "継承" します。

継承フォームへの参照は常にアクティブです。そのため、継承フォームの要素が変更されると (たとえば、ボタンスタイル)、この要素を使用する全フォームが自動的に変更されます。

テーブルフォームおよびプロジェクトフォームの両方を継承フォームとして使用できます。ただし、継承フォームに含まれる要素は、異なるデータベーステーブルでの使用に対応していなければなりません。

フォームが実行されると、オブジェクトがロードされ、次の順序で組み立てられます:

1. 継承フォームの 0ページ
2. 継承フォームの 1ページ
3. 開かれたフォームの 0ページ
4. 開かれたフォームのカレントページ

この順序により、フォームにおけるオブジェクトの [入力順](#) が決まります。

継承フォームの 0ページと 1ページだけが他のフォームに表示可能です。

継承フォームとして使用される場合、継承フォームのプロパティとフォームメソッドは使用されません。他方、継承フォームに含まれるオブジェクトのメソッドは呼び出されます。

継承フォームを設定するには、他のフォームを継承するフォームにおいて、[継承されたフォーム名](#) および [継承されたフォームテーブル](#) (テーブルフォームの場合) プロパティを設定しなければなりません。

プロジェクトフォームを継承するには [継承されたフォームテーブル](#) プロパティで <なし> を選択します (JSON の場合は " ")。

フォームの継承をやめるには、プロパティリストの [継承されたフォーム名](#) プロパティで <なし> オプション (JSONの場合は " ") を選択します。

任意のフォームで継承フォームを設定し、そのフォームを第3のフォームの継承フォームとして使用することができます。再帰的な方法で各オブジェクトが連結されます。4Dは、再帰的ループを見つけ出し (たとえば、[テーブル1]フォーム1 が [テーブル1]フォーム1 を継承フォームとして定義している、つまり自分自身を継承している場合)、フォームの連鎖を中断します。

プロパティ一覧

[フォームタイプ](#) - [フォーム名](#) - [継承されたフォームテーブル](#) - [継承されたフォーム名](#) - [ウインドウタイトル](#) - [配置を記憶](#) - [サブフォームとして公開](#) - [固定幅](#) - [最小幅](#) - [最大幅](#) - [固定高さ](#) - [最小高さ](#) - [最大高さ](#) - [印刷設定](#) - [連結メニューバー](#) - [フォームヘッダー](#) - [フォーム詳細](#) - [フォームブレーク](#) - [フォームフッター](#) - [メソッド](#) - [Pages](#)

フォームエディター

4D が提供するフォームエディターを使用して、必要とされる機能に達するまでフォームを完全にカスタマイズできます。フォームエディターでは、オブジェクトの作成や削除、操作、フォームやオブジェクトのプロパティの設定がおこなえます。

インターフェース

フォームエディターは各 JSON フォームを個別のウィンドウに表示し、ウィンドウごとにオブジェクトバーとツールバーがあります。複数のフォームを開くことができます。

表示オプション

フォームのカレントページの大部分のインターフェース要素は、表示したり非表示にしたりすることができます。

- 継承されたフォーム：継承されたフォームオブジェクト（[継承されたフォーム](#) が存在する場合）
- ページ0：[ページ0](#) のオブジェクト。このオプションで、フォームのカレントページのオブジェクトとページ0 のオブジェクトを区別することができます。
- 用紙：印刷ページの用紙境界を示す灰色の線。この要素は、[印刷用](#) タイプのフォームでのみデフォルトで表示できます。
- ルーラー：フォームエディターウィンドウのルーラー。
- マーカー：フォームのエリアを識別する出力コントロールラインとマーカー。この要素は、[リストフォーム](#) タイプのフォームでのみデフォルトで表示できます。
- マーカーラベル：マーカーラベル。これは出力コントロールラインが表示されている場合のみ有効です。この要素は、[リストフォーム](#) タイプのフォームでのみデフォルトで表示できます。
- 境界：フォームの境界。このオプションが選択されていると、アプリケーションモードで表示されるとおりに、フォームがフォームエディターに表示されます。これによりアプリケーションモードに移動しなくてもフォームを調整しやすくなります。

サイズを決めるもの、水平 マージン そして 垂直 マージン フォームプロパティ設定はフォーム境界に影響します。これらの設定を使用すると、フォーム上のオブジェクトに基づいて境界を設定できます。フォームの境界を決定する位置にオブジェクトを配置したり、サイズを変更したりすると、境界も変更されます。

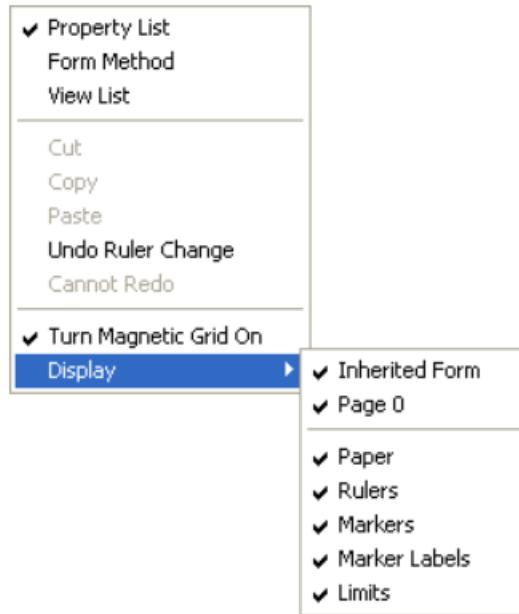
デフォルト表示

エディターでフォームを開いたとき、インターフェース要素は、以下に応じてデフォルトで表示または非表示になります：

- 環境設定で設定された 新フォームにデフォルトで表示 オプション - チェックされていないオプションはデフォルトでは表示されません。
- カレントの [フォームタイプ](#)：
 - リストフォームでは、マーカーとマーカーラベルはデフォルトで常に表示されます。
 - 用紙は "印刷用" のフォームの場合、デフォルトで表示されます。

要素の表示/非表示

フォーム メニューまたはフォームエディターのコンテキストメニューから [表示](#) を選択すると、フォームエディターのカレントウィンドウ内でいつでも要素の表示/非表示を切り替えることができます。



ルーラー

右と下にあるルーラーが、オブジェクトの配置を手助けします。これらは、[表示または非表示](#) にすることができます。

フォームメニューの ルーラー定義... を選択すると、単位を変更して、インチ、センチ、ピクセルのいずれかで表示させることができます。

ツールバー

フォームエディターのツールバーはフォームを操作・更新するための一連のツールを提供します。ウィンドウごとに固有のツールバーを持ちます。



ツールバーには以下の要素があります：

アイコン	名称	説明
▶	フォーム実行	フォームの実行をテストするために使用します。このボタンをクリックすると、4D は新しいウインドウを開き、そのコンテキストでフォームを表示します（リストフォームの場合レコードリスト、詳細フォームの場合カレントレコード）。フォームはメインプロセスで実行されます。
🖱	選択ツール	フォームオブジェクトの選択・移動・リサイズをおこないます。 注：テキストやグループボックスタイプのオブジェクトを選択すると、Enterキーを押すことで編集モードになります。
⇄	入力順	"入力順" モードに切り替わり、フォームの現在の入力順を表示・変更できます。入力順は、バッジを使用して確認することもできます。
👉	移動	"移動" モードに移行し、ウインドウ中をドラッグ & ドロップすることで素早くフォームの表示部分を移動することができます。このモードでカーソルは手の形になります。このモードは、フォームを拡大表示している時に特に便利です。
🔍	拡大	フォーム表示の拡大/縮小率を変更できます（デフォルトで100%）。“拡大/縮小” モードにするには虫眼鏡をクリックするか、拡大/縮小率バーをクリックします。この機能は前節で説明しています。
⤒	整列	このボタンには、フォーム内でオブジェクトの並びを揃えるためのメニューがリンクされています。このボタンは選択されているオブジェクトに応じて有効/無効になります。 CSS プレビューが "なし" の場合にのみ利用可能です。
⤓	均等配置	このボタンには、フォーム内でオブジェクトを均等に配置するためのメニューがリンクされています。このボタンは選択されているオブジェクトに応じて有効/無効になります。 CSS プレビューが "なし" の場合にのみ利用可能です。
⤔	レベル	このボタンには、フォーム上のオブジェクトの階層を変更するためのメニューが関連付けられています。このボタンは選択されているオブジェクトに応じて有効/無効になります。
⤕	グループ化/グループ解除	このボタンには、フォーム上の選択オブジェクトのグループ化やグループ解除をおこなうためのメニューが関連付けられています。このボタンは選択されているオブジェクトに応じて有効/無効になります。
⤖	表示とページ管理	このエリアを使用して、フォームページ間の移動やページの追加ができます。フォームページを移動するには矢印ボタンをクリックするか、または中央のエリアをクリックすると現われるメニューから表示したいページを選択します。最終ページが表示されている状態で、右矢印ボタンをクリックすると、4D はページを追加します。
⤗	CSSプレビュー	このボタンで、使用する CSSモードを選択します。
⤘	ビューマネージャー	このボタンは、ビューパレットの表示や非表示をおこないます。この機能については "オブジェクトビューを使用する" で説明しています。
⤙	バッジ表示	このボタンをクリックするたびに、すべてのタイプのフォームバッジが順に表示されます。また、このボタンには、表示するバッジタイプを直接選択できるメニューが関連付けられています。
⤚	定義済みオブジェクトライブラリ	このボタンは定義済みオブジェクトライブラリを表示します。このライブラリは定義済みのプロパティを持つオブジェクトを多数提供します。
⤛	リストボックスビルダー	このボタンは、新しいエンティティセレクション型リストボックスを作成します。

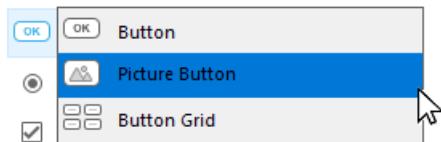
オブジェクトバーを使用する

オブジェクトバーには、4Dフォーム上で使用できるアクティブオブジェクトや非アクティブオブジェクトがすべて含まれています。一部のオブジェクトは、テーマ別にまとめられています。各テーマでは、複数の項目のなかから選択することができます。オブジェクトバーにフォーカスがある場合、キーボードのキーを使用してボタンを選択できます。以下の表で利用可能なオブジェクトグループとショートカットを示します。

ボタン	グループ	キー
	テキスト / グループボックス	T
	入力	F
	階層リスト / リストボックス	L
	コンボボックス / ドロップダウンリスト / ピクチャーポップアップメニュー	P
	ボタン / ピクチャーボタン / ボタングリッド	B
	ラジオボタン	R
	チェックボックス	C
	進捗インジケーター / ルーラー / ステッパー / スピナー	I
	四角 / 線 / 桁円	S
	スプリッター / タブコントロール	D
	プラグインエリア / サブフォーム / Webエリア / 4D Write Pro / 4D View Pro	○

任意のオブジェクトタイプを描画するには、該当するボタンを選択してから、フォーム上でそのオブジェクトを描きます。オブジェクトを作成した後でも、プロパティリストを用いてオブジェクトのタイプを変更することができます。強制的にオブジェクトを規則正しい形で描画するには、Shiftキーを押しながらオブジェクトを作成します。この場合、線は水平方向、45度、または垂直方向に引かれます。また、四角は正方形に、楕円は正円に固定されます。

そのテーマで現在選択されているオブジェクトがフォームに挿入されます。ボタンの右側をクリックすると、バリエーションメニューが表示されます:



ボタンを 2回クリックすると、フォーム上にオブジェクトを描画した後も、そのボタンが選択されたままになります（連続選択）。この機能により、同じタイプのオブジェクトを複数連続して作成しやすくなります。連続選択を解除したい場合は、別のオブジェクトやツールをクリックします。

プロパティリスト

フォームおよびフォームオブジェクトはプロパティを持ち、フォームへのアクセスやフォームの外観、およびフォーム使用時の動作が制御されます。フォームプロパティには、たとえばフォーム名、メニューバー、フォームサイズなどがあります。またオブジェクトプロパティには、たとえばオブジェクト名、オブジェクトサイズ、背景色、フォントなどがあります。

プロパティリストを使用して、フォームおよびオブジェクトプロパティを表示・変更できます。エディター上でオブジェクト選択していればそのプロパティが、オブジェクトを選択していない場合はフォームのプロパティがプロパティリストに表示されます。

プロパティリストを表示/非表示にするには、フォームメニュー、またはフォームエディターのコンテキストメニューから プロパティリストを選択します。さらに、フォームの空のエリアをダブルクリックすることでも表示させることができます。

ナビゲーションショートカット

次のショートカットを使用し、プロパティリスト内を移動することができます:

- 矢印キー ↑ ↓: あるセルから別のセルへ移動します。
- 矢印キー ← →: テーマを展開/縮小するか、入力モードに入ります。
- PgUp と PgDn: プロパティリスト内をスクロールします。

- Home と End: プロパティリストの最初または最後のセルを表示するようスクロールします。
- イベント上で Ctrl+クリック (Windows) または Command+クリック (macOS) : クリックしたイベントの最初の状態に応じて、リストの各イベントを選択/選択解除します。
- テーマレベル上で Ctrl+クリック (Windows) または Command+クリック (macOS) : リストのすべてのテーマを展開/縮小します。

フォームオブジェクトの操作

オブジェクトの追加

フォームにオブジェクトを追加する方法は複数あります:

- オブジェクトバーでオブジェクトタイプを選択し、フォームエディター上で直接それを描画する ([オブジェクトバーを使用する](#) 参照)。
- オブジェクトバーからオブジェクトをドラッグ & ドロップする。
- 定義済み [オブジェクトライブラリ](#) から選択したオブジェクトをドラッグ & ドロップあるいはコピー/ペーストする。
- 他のフォームからオブジェクトをドラッグ & ドロップする。
- エクスプローラー (フィールド) やデザインモードの他のエディター (リストやピクチャー等) からオブジェクトをドラッグ & ドロップする。

オブジェクトをフォームに配置したら、フォームエディターを使用してそのオブジェクトのプロパティを編集できます。

フォームでは 2つのタイプのオブジェクトを扱います:

- スタティックオブジェクト (線、枠、背景ピクチャー等): これらは一般的に、フォームのアピアランスやラベル、グラフィックインターフェースを設定するために使用されます。これらはフォームエディターのオブジェクトバーから利用できます。プロパティリストを使用して、これらのグラフィック属性 (サイズ、カラー、フォント等) やリサイズオプションも指定できます。アクティブオブジェクトと異なり、スタティックオブジェクトには変数や式が割り当てられません。しかし、スタティックオブジェクトにダイナミックオブジェクトを挿入することは可能です。
- アクティブオブジェクト: この種のオブジェクトはインターフェース中でタスクや機能を実行します。フィールド、ボタン、リストボックスなど様々な種類があります。各アクティブオブジェクトにはフィールドまたは変数が割り当てられます。

オブジェクトの選択

オブジェクトの操作 (線幅やフォントの変更など) をおこなう前に、対象のオブジェクトを選択する必要があります。

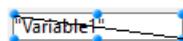
ツールバーを使用してオブジェクトを選択するには:

1. ツールバーの矢印ツールをクリックします。



マウスカーソルをフォームエリアに移動すると、カーソルは標準の矢印の形をしたポインターに変わります。

2. 選択したいオブジェクトをクリックします。サイズ変更ハンドルが表示され、オブジェクトが選択されたことを表わします。



プロパティリストを使用してオブジェクトを選択するには:

1. プロパティリストの一番上にあるオブジェクトリストドロップダウンリストからオブジェクト名を選択します。

この方法では、他のオブジェクトの下に隠れているオブジェクトや、カレントウィンドウの表示領域外に置かれているオブジェクトを選択することができます。オブジェクトの選択を解除するには、オブジェクト境界の外側をクリックするか、またはオブジェクト上で Shift+クリック します。

"デザインモードを検索" の結果ウィンドウでオブジェクトをダブルクリックして選択することもできます。

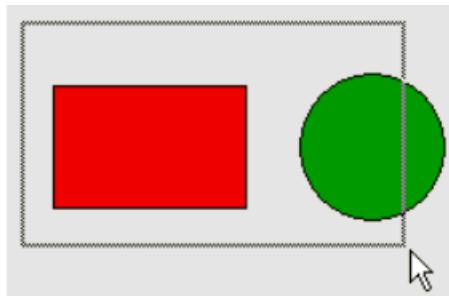
複数オブジェクトの選択

複数のフォームオブジェクトに関して同じ操作を適用したい場合があります。たとえば、オブジェクトの移動や整列、外観の変更をおこなう場合などです。4D では一度に複数のオブジェクトを選択することができます。複数のオブジェクトを選択する方法はいくつかあります:

- 編集メニューから すべてを選択 を選択して、すべてのオブジェクトを選択する。
- オブジェクト上で右クリックし、コンテキストメニューから 同じ種類のオブジェクトを選択 コマンドを選択する。
- Shiftキーを押しながら、選択したいオブジェクトをクリックする。

- 選択したいオブジェクトグループの外側から各オブジェクトを囲むようにマーキー（選択矩形とも呼ばれます）を描画する。マウスボタンを離すと、マーキー内及びマーキーに重なるオブジェクトが選択されます。
- Altキー（Windows）または、Optionキー（macOS）を押しながら、マーキーを描画します。マーキーに完全に囲まれたオブジェクトが選択されます。

次の図はマーキーが描画され、2つのオブジェクトが選択されている様子を示しています：



一連の選択オブジェクトから任意のオブジェクトを除外するには、Shiftキーを押しながらそのオブジェクトをクリックします。この場合、他のオブジェクトは選択されたままになります。選択されているオブジェクトをすべて選択解除するには、いずれのオブジェクトの境界にもかからない場所をクリックします。

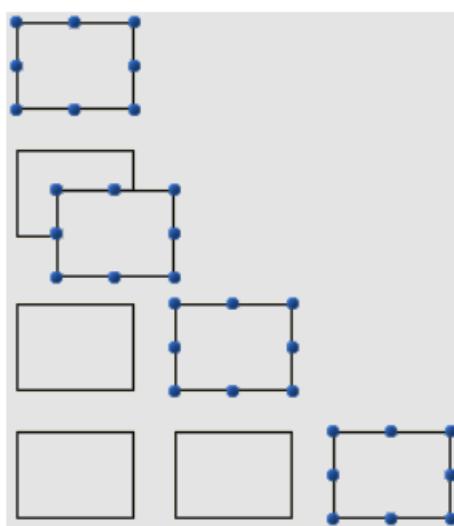
オブジェクトの複製

アクティボブジェクトを含む任意のオブジェクトをフォーム上で複製できます。アクティボブジェクトのコピーはオブジェクト名を除き、変数名、型、標準アクション、表示フォーマット、オブジェクトメソッドなどすべてのプロパティが保持されます。

ツールパレットの複製ツールを使用してオブジェクトを直接複製するか、"行列を指定して複製" ダイアログボックスでオブジェクトを複数一気に作成できます。このダイアログでは、2つのコピー間の間隔も指定できます。

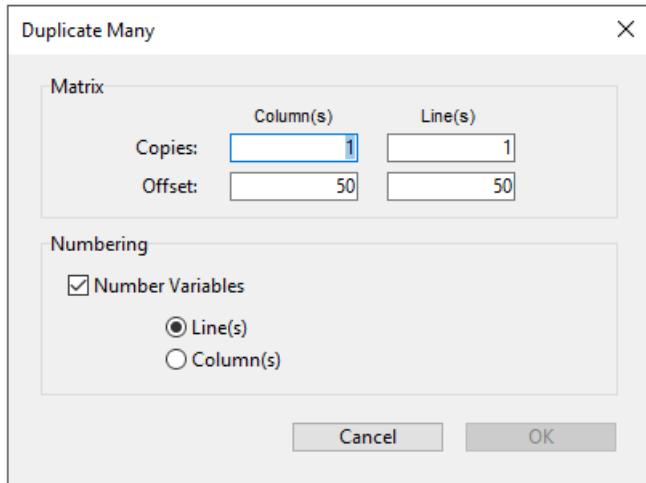
オブジェクトを複製するには：

- 複製したいオブジェクトを選択します。
- 編集メニューから 複製 を選択します。4D は選択されたオブジェクトのコピーを作成し、オリジナルオブジェクトの基点から右下に配置します。複製されたコピーはオリジナルの前面におかれます。
- コピーを適切な場所に配置します。ここで複製メニュー項目を再び選択すると、4D はもう一つコピーを作成し、最初のコピーと元のオブジェクトとの同じ距離同じ方向に配置します。これを活かし、オブジェクトのコピーのあるライン上に配置する必要がある場合は、以下の手順でおこないます。元のオブジェクトを複製し、そのコピーをフォーム内の別の場所に移動させてから、コピーを複製します。1つ目のコピーと元のオブジェクトの位置関係を再現する形で、2つ目のコピーも 1つ目のコピーに対して、自動的に配置されます。後続のコピーも、それぞれのコピー元オブジェクトと同じ位置関係に配置されます。以下の図は、この相対的なコピーの配置が動作する様子を示しています：



行列を指定して複製

"行列を指定して複製" ダイアログボックスは、オブジェクトメニューから 行列を指定して複製... コマンドを選択すると表示されます。



- 上のエリアには、作成したいオブジェクトの列数と行数を入力します。

たとえば、3列 2行のオブジェクトを作成したい場合、列に 3 を、行に 2 を入力します。横に 3つの新しいコピーを作成したい場合は、列欄に 4 を入力し、行はデフォルトの 1 のままにします。

- 列と行それぞれに、コピー間のオフセットを指定できます。

値はポイント単位で指定します。オフセットは、元のオブジェクトに対して相対的に、コピー毎に適用されます。

たとえば、元のオブジェクトの高さが 50 ポイントである場合、オブジェクトごとに 20 ポイント縦オフセットするには、列の "オフセット" エリアに 70 を入力します。

- 格子状に変数を作成したい場合、変数に番号設定 オプションを選択し、番号を振る方向を行または列から選択します。選択したオブジェクトが変数の場合にのみ、このオプションは有効になります。詳細は [デザインリファレンスで「グリッド上にオブジェクト作成」](#) を参照ください。

オブジェクトの移動

テンプレートで作成されたフィールドやオブジェクトを含め、フォーム上のグラフィックやアクティブオブジェクトはすべて移動可能です。オブジェクトを移動するには、次のような方法があります：

- オブジェクトをドラッグして移動する。
- 矢印キーを使用して、オブジェクトを 1ピクセルずつ移動する。
- Shiftキーと矢印キーを使用して、オブジェクトを 1ステップずつ移動する（デフォルトで 1ステップ=20ピクセル）。

選択したオブジェクトのドラッグを開始すると、ハンドルが消えます。4D はルーラーにオブジェクトの座標を示すマーカーを表示するので、適切な位置にオブジェクトを配置することができます。このとき、ハンドルをドラッグしないようにしてください。ハンドルをドラッグすると、オブジェクトのサイズが変更されます。Shiftキーを押しながらドラッグすると、制約付きの移動になります。

[マグネティックグリッド](#) が有効な場合、グリッドに吸着するようにオブジェクトが移動されます。

オブジェクトを 1ピクセルずつ移動するには：

- 移動したいオブジェクトを選択し、キーボード上の矢印キーを使用してオブジェクトを移動します。矢印キーを押すたびに、矢印の方向へオブジェクトが 1ピクセルずつ移動します。

オブジェクトを 1ステップずつ移動するには：

- 移動したいオブジェクトを選択し、Shiftキーを押しながらキーボード上の矢印キーを使用して、オブジェクトを移動します。デフォルトで 1ステップにつき 20ピクセル移動します。このピクセル数は、環境設定のフォームページで変更できます。

オブジェクトのグループ化

4D ではオブジェクトをグループ化して、そのグループを一つのオブジェクトとして選択・移動・変更することができます。グループ化されたオブジェクト同士の相対位置は保持されます。一般的には、フィールドとそのラベル、透明ボタンとそのアイコン等をグループ化するでしょう。

グループの大きさを変更すると、そのグループ内の全オブジェクトのサイズが同じ比率で変更されます（テキストエリアは除きます。テキストエリアのサイズは、そのフォントサイズに合わせて変更されます）。

グループ化を解除すると、再び個々にオブジェクトを扱えるようになります。

グループ化されたアクティブオブジェクトのプロパティやメソッドにアクセスするには、グループ化を解除しなければなりません。しかし、グループに属するオブジェ

クトを、グループ化を解除せずに選択することは可能です。これには、オブジェクトを Ctrl+クリック (Windows) または Command+クリック (macOS) します (グループはあらかじめ選択されている必要があります)。

グループ化はフォームエディター上でのみ意味を持ちます。フォームの実行中は、グループ化されたすべてのオブジェクトが、グループ化されていないのと同じに動作します。

異なるビューに属するオブジェクトをグループ化することはできず、カレントビューに属するオブジェクトのみをグループ化することができます ([オブジェクトビュー 参照](#))。

オブジェクトをグループ化するには:

1. グループ化したいオブジェクトを選択します。
2. オブジェクトメニューから グループ化 を選択します。

OR

フォームエディターのツールバーでグループ化ボタンをクリックします。



4D は、新たにグループ化されたオブジェクトの境界をハンドルで表わします。グループ内の各オブジェクトの境界にはハンドルが表示されません。これ以降、グループ化されたオブジェクトを編集すると、グループを構成する全オブジェクトが変更されます。

オブジェクトのグループ化を解除するには:

1. グループ化を解除したいグループオブジェクトを選択します。
2. オブジェクト メニューから グループ化解除 を選択します。

OR

フォームエディターのツールバーで グループ化解除 ボタン (グループ化 ボタンのサブ項目) をクリックします。

グループ化解除 が選択不可の場合、選択したオブジェクトはグループに属していないことを意味します。

4D は個々のオブジェクトの境界をハンドルで表わします。

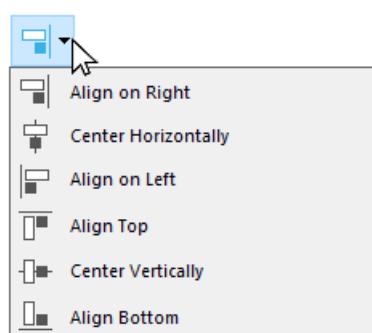
オブジェクトの整列

フォーム上のオブジェクトを互いに整列させたり、または透明グリッドを用いて揃えることができます。

- オブジェクト同士を整列させる場合、オブジェクトの上端、下端、側面で揃えたり、または別のオブジェクトの縦や横の中心線に沿って揃えることができます。整列ツールを使用して選択オブジェクトを直接揃えたり、または整列アシスタントを用いてさらに詳細な整列を適用することも可能です。整列アシスタントを使用した場合、たとえば、整列の基準となるオブジェクトを選択したり、整列を適用する前にフォーム上の整列状態をプレビューできるようになります。
- 透明グリッドを使用すると、各オブジェクトの位置を手動で揃えることができます。その際、移動中のオブジェクトが別のオブジェクトに接近すると、"目に見える" 位置ガイドとして点線が表示され、これに基づいて各オブジェクトの整列を実行することができます。

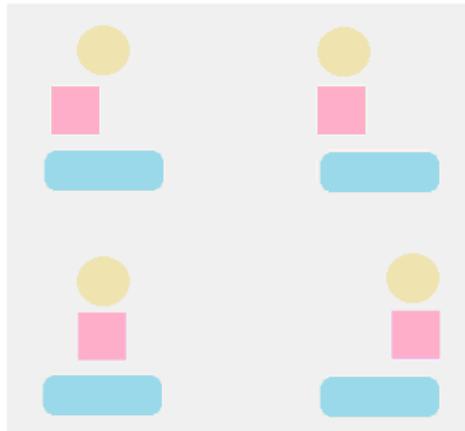
即時整列ツールを使用する

ツールバーの整列ツール、またはオブジェクトメニューの整列サブメニューを使用して、選択したオブジェクトを瞬時に揃えることができます。



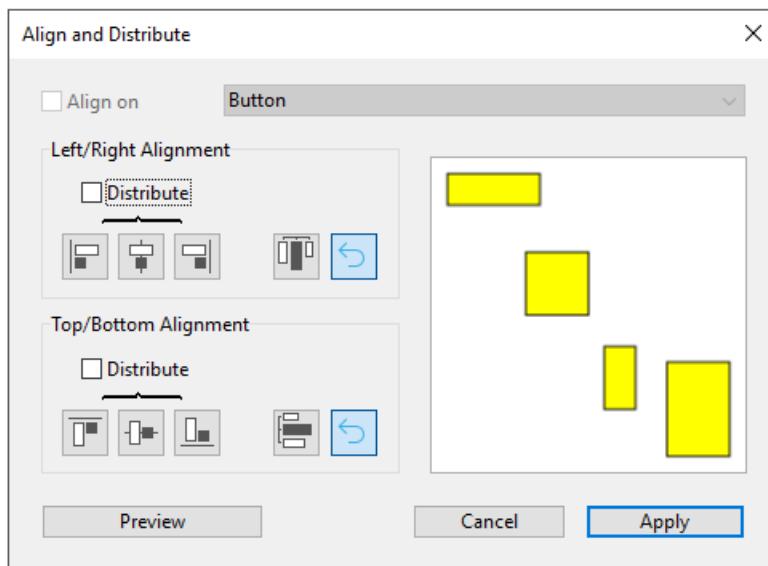
4D がオブジェクトを揃える場合、選択オブジェクトのうち 1つを定位置に置いたまま、そのオブジェクトを基準に残りのオブジェクトを整列させます。この基準オブジェクトが "アンカー" となります。整列をおこなう方向で最も離れた位置にあるオブジェクトがアンカーとして使用され、他のオブジェクトはこのオブジェクトに合わせられます。たとえば、一連のオブジェクトに対して右揃えを実行したい場合、一番右側に位置するオブジェクトがアンカーとして使用されま

す。次の図は整列なし、左揃え、縦中央揃え、右揃えの状態を示しています:



整列アシスタントを使用する

整列アシスタントを使用すると、オブジェクトに関するあらゆるタイプの整列や均等配置を実行することができます。



このダイアログボックスを表示するには、揃えたいオブジェクトを選択し、オブジェクト メニュー、またはエディターのコンテキストメニューの 整列 サブメニューから 整列... コマンドを選択します。

- "左/右整列" や "上/下整列" エリアで、実行しようとする整列に対応する整列アイコンをクリックします。

見本エリアには、選択結果が表示されます。

- 標準のアンカー方式による整列を実行するには、プレビュー または 適用 をクリックします。

この場合、整列をおこなう方向で最も離れた位置にあるオブジェクトがアンカーとして使用され、他のオブジェクトはこのオブジェクトに合わせられます。たとえば、一連のオブジェクトに対して右揃えを実行したい場合、一番右側に位置するオブジェクトがアンカーとして使用されます。

または:

特定のオブジェクトを基準にオブジェクトを揃えるには、整列 オプションを選択し、整列基準としたいオブジェクトを一覧から選択します。この場合、基準オブジェクトの位置は変わりません。

プレビュー ボタンをクリックすると、整列の結果をプレビューすることができます。するとフォームエディター上のオブジェクトは見かけ上整列しますが、ダイアログ ボックスが表示されたままなので、この整列のキャンセルや適用をおこなうことができます。

整列アシスタントを使用すると、1回の操作でオブジェクトの整列や均等配置をおこなえます。オブジェクトを均等配置する方法についての詳細は、[オブジェクトの均等配置](#) を参照ください。

マグネティックグリッドを使用する

フォームエディターには仮想的なマグネティックグリッド機能があります。この機能は、フォーム上でオブジェクトの配置や整列をおこなう際に役立ちます。オブ

ジェクトのマグネティック整列は、オブジェクト同士の相対位置に基づいておこなわれます。マグネットイックグリッドは、2つ以上のオブジェクトがフォーム上に存在する場合のみ使用可能です。

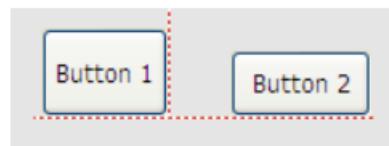
この機能は次のように作用します。フォーム上のあるオブジェクトを移動する際、4D は他のフォームオブジェクトとの相対的な位置関係に基づき、このオブジェクトの位置候補を示します。次のような場合に候補が示されます：

- 水平方向に 2つのオブジェクトの端または中央が同じ位置になる場合。
- 垂直方向に 2つのオブジェクトの端が同じ位置になる場合。

この状況になると、4D はその位置にオブジェクトを配置し、そこが候補位置であることを示す赤いラインを表示します：



オブジェクトを均等配置する場合、4D はインターフェース標準に基づいてオブジェクト間の距離を提示します。マグネットイック整列と同様に、配置が決定した時点で、その間隔が赤いラインで表わされます。



この処理は、あらゆるタイプのフォームオブジェクトに対して適用されます。フォーム メニューまたはエディターのコンテキストメニューの マグネットイックグリッド コマンドを使用して、マグネットイック機能をいつでも有効または無効に設定できます。また、環境設定 の フォーム ページにおいて、この機能をデフォルトで有効に設定しておくことも可能です（デフォルトで自動揃えを有効にする オプション）。Ctrlキー (Windows) または Controlキー (macOS) を押してオブジェクトを選択すると、手動でマグネットイックグリッドが有効/無効に設定されます。

手動でオブジェクトサイズを変更する場合も、このマグネットイックグリッドの影響を受けます。

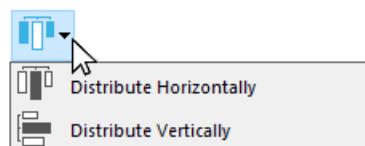
オブジェクトの均等配置

各オブジェクトが同じ間隔で配置されるように、オブジェクトを均等配置することができます。これには、ツールパレットまたは整列アシスタントの均等配置ツールを用います。整列アシスタントを使用すると、1回の操作でオブジェクトの整列や均等配置をおこなえます。

マグネットイックグリッド が有効の場合、オブジェクトを手動で動かすと、均等配置のためガイドが表示されます。

同じ間隔を空けてオブジェクトを配置するには：

- 3つ以上のオブジェクトを選択し、希望する均等配置ツールをクリックします。
- 適用したい均等配置に対応する整列ツールをツールバー上で選択します。



OR

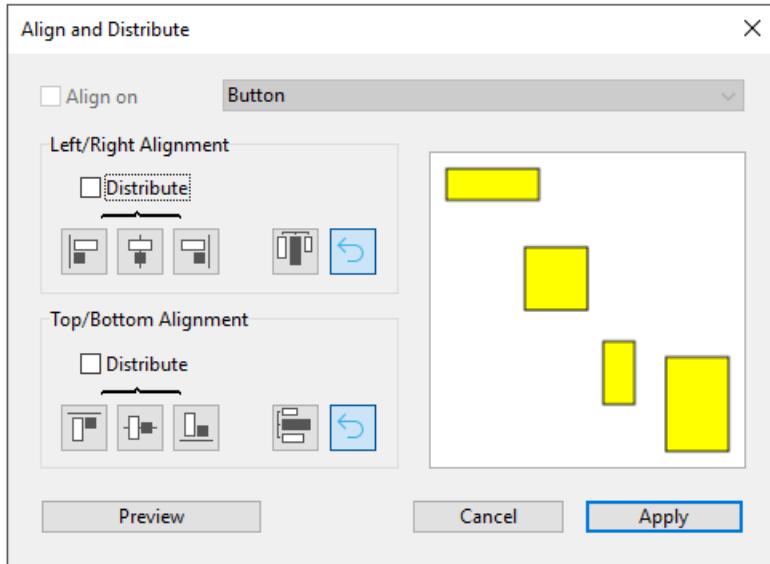
オブジェクト メニュー、またはエディターのコンテキストメニューの 整列 サブメニューから均等揃えメニュー命令を選択します。

4D は各オブジェクトを均等に配置します。各オブジェクトの中心までの間隔、および隣接する 2つのオブジェクトの間隔のうち最も広い間隔が基準として用いられます。

"整列と均等配置" ダイアログボックスを用いてオブジェクトを均等に配置するには：

- 均等配置したいオブジェクトを選択します。
- オブジェクト メニュー、またはエディターのコンテキストメニューの 整列 サブメニューから 整列... コマンドを選択します。

以下のダイアログボックスが表示されます：



3. "左/右整列" や "上/下整列" エリアで、標準の均等配置アイコンをクリックします:



(標準の横均等揃えアイコン)

見本エリアには、選択結果が表示されます。

4. 標準の均等配置を実行するには、プレビュー または 適用 をクリックします。

この場合、4D は標準の均等配置を実行し、オブジェクトは等間隔で配置されます。

または:

特定の均等配置を実行するには、均等配置 オプションを選択します (たとえば各オブジェクトの右辺までの距離をもとにオブジェクトを均等に配置したい場合)。このオプションはスイッチのように機能します。均等配置チェックボックスが選択されていると、このオプションの下にあるアイコンは異なる動作をおこないます:

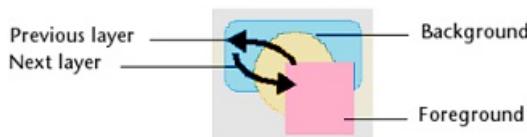
- 左/右整列の場合、各アイコンは次の均等配置に対応します: 選択オブジェクトの左辺、中央 (横)、右辺で均等に揃えます。
- 上/下整列の場合、各アイコンは次の均等配置に対応します: 選択オブジェクトの上辺、中央 (縦)、下辺で均等に揃えます。

プレビュー ボタンをクリックすると、この設定による結果をプレビューすることができます。この表示はフォームエディター上で実行されますが、ダイアログボックスは前面に表示されたままであります。この後、変更の キャンセル または 適用 をおこなうことができます。

このダイアログボックスでは、整列と均等配置を合わせて実行することができます。整列に関する詳細は [オブジェクトの整列](#) を参照ください。

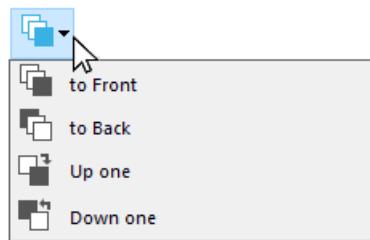
オブジェクトを重ねる

フォーム上で他のオブジェクトを隠しているオブジェクトの配置を調整する必要が生じることもあります。たとえば、フォーム上のフィールドの背景にグラフィックを配置したい場合です。4D には 前面へ、背面へ、一つ上のレベルへ、一つ下のレベルへ という 4つのメニュー項目があり、これらのコマンドを使用してフォーム上のオブジェクトを "重ねる (レイヤー)" ることができます。この重なりは、デフォルトの入力順も規定します (下の データの入力順 参照)。次の図は、他のオブジェクトの前面/背面に置かれたオブジェクトを示しています:



選択したオブジェクトのレベルを変更するには、以下のいずれかの作業をおこないます:

- オブジェクトメニューから 前面へ、背面へ、レベルを上げる、レベルを下げる のいずれかを選択します。
- エディターのコンテキストメニューの レベルに移動 サブメニュー内のコマンドのいずれかを選択します。
- ツールバーのレベル管理ボタンに割り当てられたコマンドのいずれかを選択します。



複数のオブジェクトが重なっている場合、Ctrl+Shift+クリック / Command+Shift+クリック ショートカットを使用して、クリックするたびに下のレイヤーにあるオブジェクトを選択できます。

レベルの順序を考えるにあたって、4D は常に背面から全面へと進みます。したがって、前・次で表現した場合、"前レベル" は 1つ背面のレベルのことになります。"次レベルは" 1つ前面のレベルのことです。

データの入力順

データ入力順とは、入力フォームで Tabキーや 改行キーを押したときに、フィールドやサブフォーム、その他のアクティブオブジェクトが選択される順番のことです。Shift+Tab や Shift+改行キーを押すことで、フォーム内を逆方向 (逆の入力順) に移動することもできます。

入力順は、`FORM SET ENTRY ORDER` および `FORM GET ENTRY ORDER` コマンドを使用することでランタイムで変更することができます。

フォーカス可プロパティをサポートするすべてのオブジェクトが、デフォルトでデータ入力順序に含まれます。

JSONフォームの入力順序の設定は、`entryOrder` プロパティで行います。

独自の入力順を指定しない場合、4D はオブジェクトの階層に従い、"背面から前面" へ向けてデフォルトの入力順を決定します。したがって、標準の入力順はフォーム上でのオブジェクトの作成順になります。

フォームでは度々、独自の入力順が必要になります。たとえば、次の図ではフォームの作成後に、住所に関連するフィールドが追加されています。この結果、標準の入力順が意味をなさなくなり、扱いづらい順番でデータを入力しなければなりません：

Last Name :	[People]Last Name
First Name :	[People]First Name
Full address :	[People]Address
Telephone :	[People]Phone
Company :	[People]Company
Hire date :	[People]Hire_

このようなケースでは、独自のデータ入力順を指定すると、より理にかなった順序でデータを入力できるようになります：

Last Name :	[People]Last Name
First Name :	[People]First Name
Full address :	[People]Address
Telephone :	[People]Phone
Company :	[People]Company
Hire date :	[People]Hire_

データ入力順の表示と変更

"入力順" バッジまたは "入力順" モードを使用して、現在の入力順を表示することができます。しかし、入力順を変更するには、"入力順" モードを使用しなければなりません。

この節では "入力順" モードを用いて、入力順の表示と変更をおこなう方法について説明します。バッジを用いた入力順の表示についての詳細は、[バッジを使用する](#) を参照ください。

入力順モードに切り替え、入力順を変更するには:

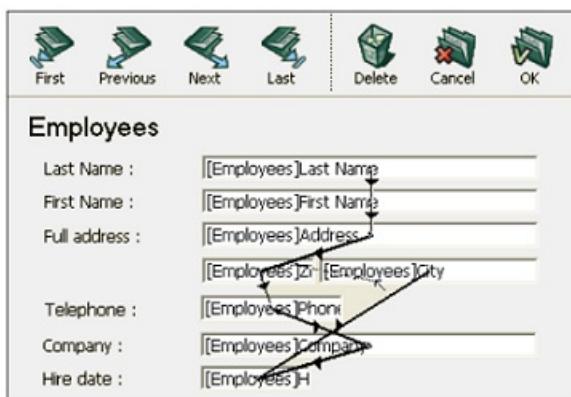
1. フォーム メニューから 入力順 を選択するか、ツールバーの入力順ボタンをクリックします:



ポインターが入力順ポインターに変わり、4D はフォーム上に線を引いて、データ入力時にオブジェクトが選択される順序を示します。

ツールパレット上の他のツールをクリックするまでは、入力順序の表示と変更操作しかおこなえません。

2. データ入力順を変更するには、フォームオブジェクト上にポインターを置き、マウスボタンを押したまま、次の入力順に設定したいオブジェクトまでポインターをドラッグします。



これに応じて、4D はデータ入力順を調整します。

3. 入力順を設定したいだけ、ステップ2 を繰り返します。
4. 入力順の設定が終了したら、ツールバーの他のツールをクリックするか、フォーム メニューから 入力順 を選択します。

4Dは、フォームエディターの通常操作に戻ります。

フォームのカレントページの入力順だけが表示されます。フォームのページ0 や継承フォームに入力可オブジェクトが含まれている場合、デフォルトの入力順は次のようにになります: 継承フォームのページ0 のオブジェクト→ 継承フォームのページ1 のオブジェクト→ 開かれているフォームのページ0 のオブジェクト→ 開かれているフォームのカレントページのオブジェクト。

データ入力グループを使用する

入力順序を変更する際に、フォーム上のオブジェクトグループを選択し、そのグループ内のオブジェクトに対して標準の入力順序を適用することも可能です。これにより、フィールドがグループや列に分かれているフォーム上で、データ入力順序を簡単に設定することができます。

データ入力グループを作成するには:

1. フォーム メニューから 入力順 を選択するか、ツールバーの入力順ボタンをクリックします。
2. データ入力用のグループに指定したいオブジェクトの周囲をマーキーで囲みます。

マウスボタンを放すと、マーキーに囲まれているオブジェクトや、その矩形に接しているオブジェクトが標準入力順に設定されます。それ以外のオブジェクトのデータ入力順は、必要に応じて調整されます。

フィールドを入力順から除外する

デフォルトでは、すべてのフォーカス可オブジェクトが入力順に組み込まれています。任意のオブジェクトを入力順から除外するには:

1. 入力順モードに切り替えます。
2. オブジェクト上で Shift+クリック します。
または

3. オブジェクト上で右クリックし、コンテキストメニューから 入力順から削除する を選択します。

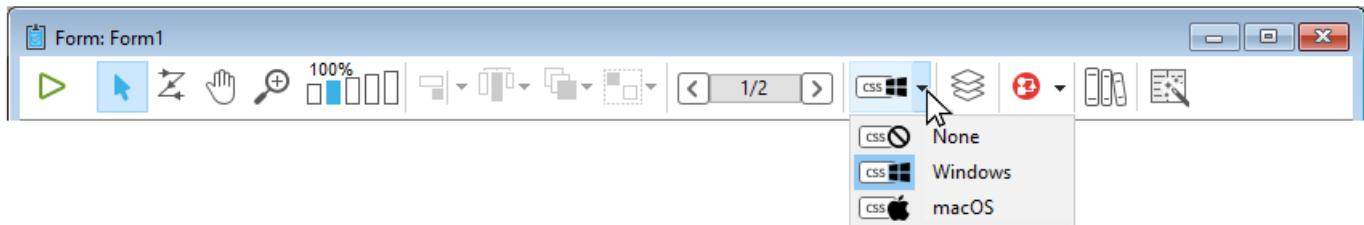
CSSプレビュー

フォームエディターでは、CSSの値を適用した状態、または適用しない状態でフォームを表示することができます。

スタイルシート が定義されている場合、フォーム（継承フォームとサブフォームを含む）はデフォルトで、現在の OS の CSSプレビューモードで開かれます。

CSSプレビューモードの選択

フォームエディターのツールバーには、スタイル付きオブジェクトを表示するための CSSボタンがあります：



メニューから、以下のプレビューモードのいずれかを選択します：

ツールバーアイコン	CSSプレビューモード	説明
	なし	CSS の値はフォームに適用されず、CSS の値やアイコンはプロパティリストに表示されません。
	Windows	Windowsプラットフォーム用の CSS値がフォームに適用されます。プロパティリストに CSSの値とアイコンが表示されます。
	macOS	macOSプラットフォーム用の CSS値がフォームに適用されます。プロパティリストに CSSの値とアイコンが表示されます。

オブジェクトに対して大きすぎるフォントサイズがスタイルシートまたは JSON で定義されている場合、オブジェクトは自動的にフォントに合わせてレンダリングされますが、オブジェクトのサイズは変更されません。

CSSプレビューモードは、[JSON vs スタイルシート](#) の項で定義した、スタイルシートと JSON属性に適用される優先順位を反映します。

CSSプレビューモードを選択すると、オブジェクトは自動的にスタイルシートで定義されたスタイル（あれば）で表示されます。

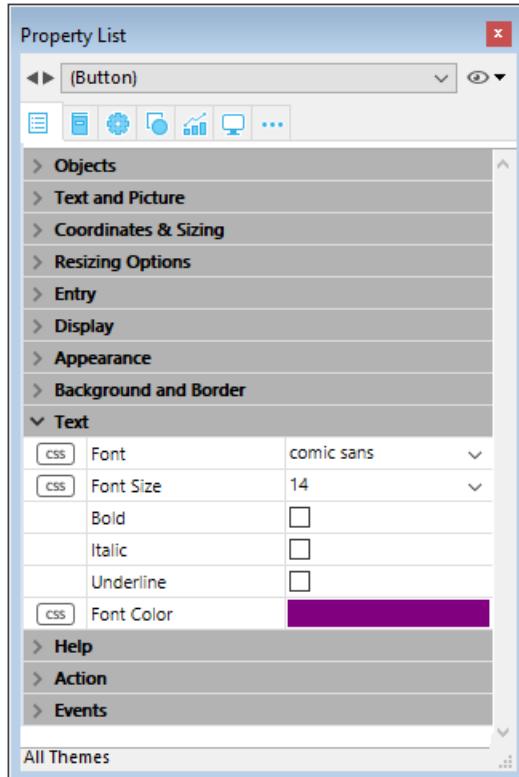
オブジェクトをコピーまたは複製すると、CSS参照（あれば）と JSON値のみがコピーされます。

プロパティリストの CSSサポート

CSSプレビューモードでは、スタイルシートで属性値が定義されている場合、その属性名が表示され、その横に CSSアイコンが表示されます。たとえば、このスタイルシートで定義されている属性値は：

```
.myButton {  
font-family: comic sans;  
font-size: 14;  
stroke: #800080;  
}
```

プロパティリストに CSSアイコンとともに表示されます：



スタイルシートで定義された属性値は、JSONフォームの記述でオーバーライドすることができます（ただし、CSSに `!important` 宣言が含まれている場合は除きます。後述参照）。この場合、プロパティリストでは、JSONフォームの値が 太字 で表示されます。Ctrl+クリック (Windows) または Command+クリック (macOs) のショートカットで、値をスタイルシートの定義に戻すことができます。

グループ、グループ内のオブジェクト、または複数オブジェクトの選択範囲内のオブジェクトに対して、`!important` 宣言とともに属性が定義されている場合、その属性値はロックされ、プロパティリストで変更することはできません。

プロパティリスト CSSアイコン

アイコン	説明
<code>CSS</code>	属性値がスタイルシートで定義されていることを示します
<code>CSS!</code>	属性値がスタイルシートで <code>!important</code> 宣言とともに定義されていることを示します
...	グループまたは複数のオブジェクトの選択項目のうち、少なくとも 1つのオブジェクトについて、スタイルシートで定義された属性値が他のオブジェクトと異なる場合に表示されます。

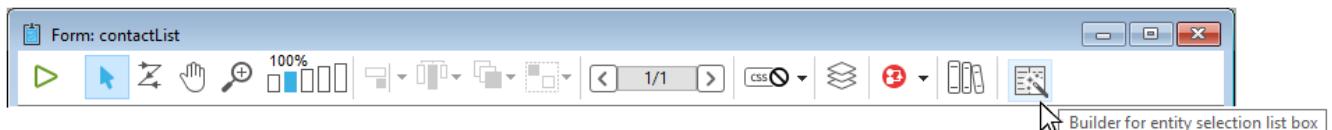
リストボックスビルダー

リストボックスビルダー を使用して、エンティティセレクション型リストボックスを素早く作成することができます。作成したリストボックスは、すぐに使用することも、フォームエディターで編集することもできます。

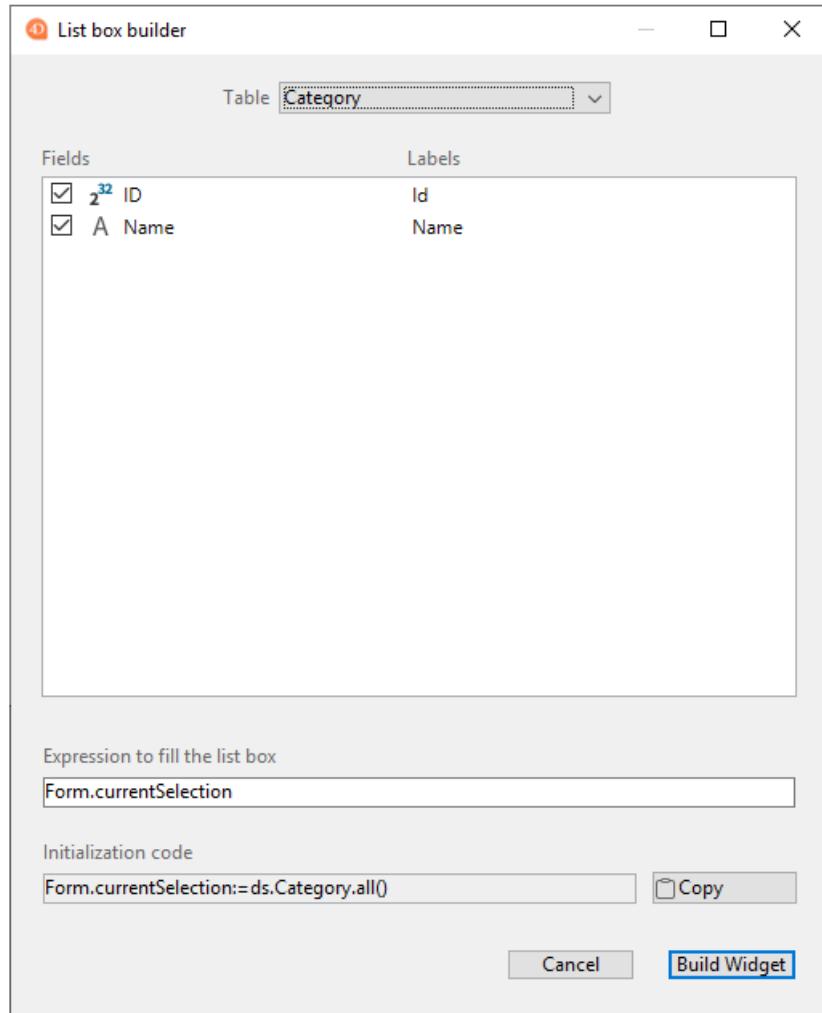
リストボックスビルダーでは、いくつかの簡単な操作で、エンティティセレクション型リストボックスの作成と入力ができます。

リストボックスビルダーを使用する

1. フォームエディターツールバーのリストボックスビルダーアイコンをクリックします:



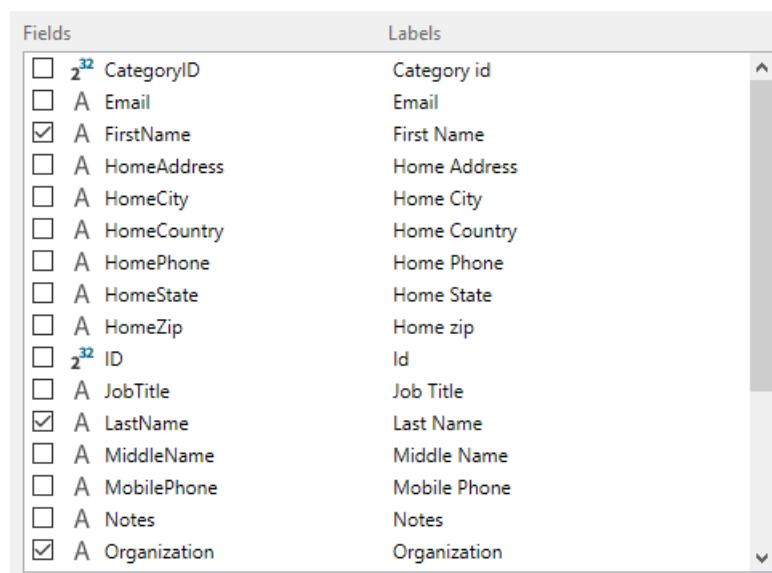
リストボックスビルダーが表示されます:



2. テーブル ドロップダウンリストからテーブルを選択します:



3. フィールド エリアで、リストボックスに表示するフィールドを選択します:



デフォルトでは、すべてのフィールドが選択されています。フィールドは個別に選択/選択解除するか、Ctrl+クリック (Windows) または Cmd+クリック (macOS) で一括に選択/選択解除することができます。

また、フィールドをドラッグ & ドロップすることで、フィールドの順番を変更することができます。

4. リストボックスをエンティティセレクションと紐づけるための式があらかじめ入力されています:

Expression to fill the list box

`Form.currentSelection`

この式は必要に応じて変更できます。

5. コピー ボタンをクリックすると、全レコードをメモリに読み込む式が初期化用にコピーされます。

Initialization code

```
Form.currentSelection:= ds.Contact.all()
```

 Copy

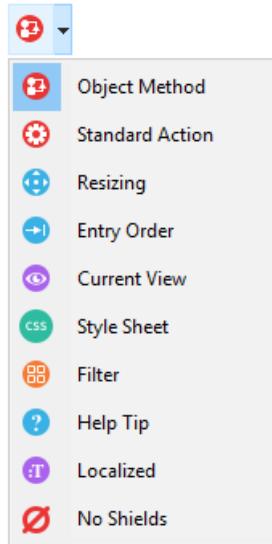
6. ウィジェットをビルト ボタンをクリックすると、リストボックスが作成されます。

Build widget

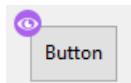
結果のリストボックスです:

バッジ

フォームエディターではバッジを使用してオブジェクトプロパティの表示を容易にできます。バッジは、フォームのツールバーで選択します:



この機能は次のように動作します：各バッジは特定のプロパティに対応しています（たとえば、カレントビューは、当該オブジェクトがカレントビュー内にあることを示します）。バッジを有効にすると、4D はバッジとして選択されたプロパティが割り当てられているフォームオブジェクトの左上に小さなアイコン（バッジ）を表示します。



バッジを使用する

バッジを有効にするには、希望するバッジが選択されるまでツールバーの バッジ ボタンをクリックします。また、ボタンの右側をクリックして表示されるメニューから、バッジの種類を選択することもできます。

バッジを表示たくない場合は、バッジなし を選択します。

アプリケーション環境設定のフォームページで、デフォルトで表示するバッジを設定できます。

各バッジの説明

各バッジの説明は以下の通りです:

アイコン	名称	表示
🔗	オブジェクトメソッド	オブジェクトメソッドが割り当てられたオブジェクト
⚙️	標準アクション	標準アクションが割り当てられたオブジェクト
⌚	サイズ変更	リサイズプロパティが 1つ以上割り当てられたオブジェクトについて、カレントプロパティの組み合わせを表します
➡️	入力順	入力可能なオブジェクトの入力順を表示します
👁️	カレントビュー	カレントビュー内にあるオブジェクト
CSS	スタイルシート	1つ以上の属性値がスタイルシートにより上書きされたオブジェクト
FilterWhere	フィルター	入力フィルターが割り当てられた入力可オブジェクト
❓	ヘルプTips	ヘルプTips が割り当てられたオブジェクト
Localization	ローカライズ済み	ラベルに参照が割り当てられたオブジェクト (":"で始まるラベル)。参照はリソース (STR#) または XLIFFタイプ
∅	バッジなし	バッジは表示されません

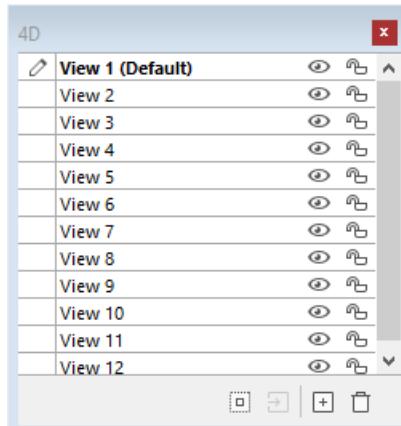
オブジェクトビュー

フォームエディターでは、必要に応じて表示/非表示が可能な異なるビューにそれぞれオブジェクトを配置することで、複雑なフォーム作成が容易になります。

たとえば、タイプごと (フィールド、変数、スタティックオブジェクト等) にオブジェクトを異なるビューに分けることができます。サブフォームやプラグインエリアを含むすべてのタイプのオブジェクトをビューに含めることができます。

フォームごとのビューの数に制限はありません。必要なだけ、ビューを作成することができます。それぞれのビューごとに、表示/非表示とロックが切り換えられます。

ビューの管理はビューパレットを使用しておこないます。



ビューパレットの表示

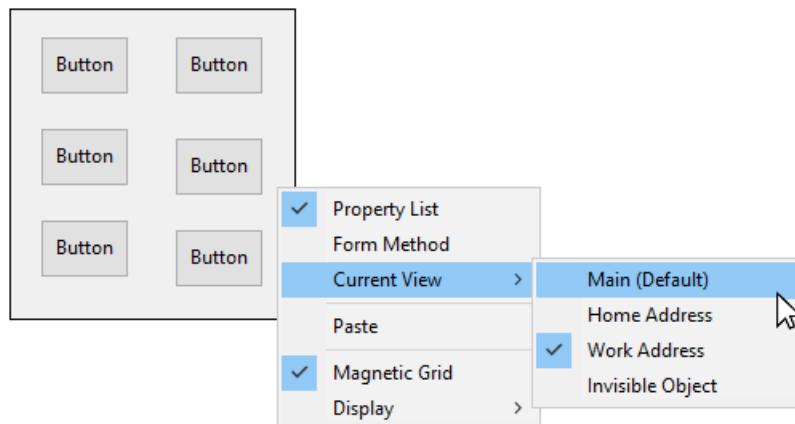
ビューパレットを表示するには、次の 3つの方法があります:

- ツールバー: フォームエディターのツールバーにあるビュー ボタンをクリックする。(1つ以上のオブジェクトがデフォルトビュー以外のビューに属している場

合、このアイコンは塗り潰し表示されます)。

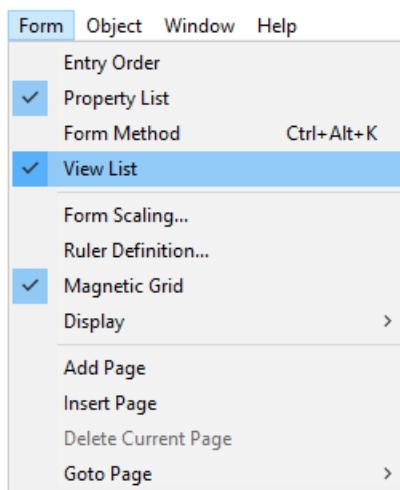
デフォルトビューのみ	追加のビューあり

- コンテキストメニュー (フォームまたはオブジェクト): フォームエディターまたはオブジェクト上で右クリックし、カレントビューを選択する。



選択中のビューにはチェックマークが付いています (例: 上の画像では "Work Address" ビューが選択中です)

- フォームメニュー: フォームメニューからビューリストを選択する。



ビューを使い始める前に

ビューを使用する前に知っておくべき重要なことをいくつか紹介します:

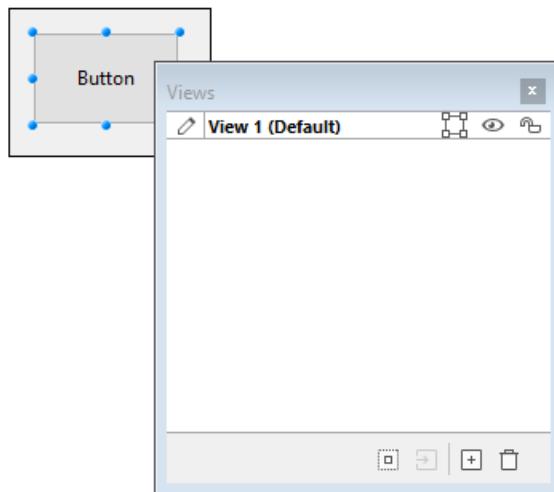
- 利用のコンテキスト: ビューは純粋に表示ツールであり、フォームエディター上でのみ効果があります。プログラムからビューにアクセスしたり、アプリケーションモードで使用したりすることはできません。
- ビューとページ: あるビューには異なるフォームページ上のオブジェクトを含めることができます。ビューの設定にかかわらず、カレントページ (およびページ0) のオブジェクトのみが表示されます。
- ビューとレベル: ビューはオブジェクトレベルから独立しています。異なるビューの間で表示上の階層はありません。
- ビューとグループ: カレントビューに属するオブジェクト同士のみをグループ化できます。
- カレント/デフォルトビュー: デフォルトビューはフォームの最初のビューで、削除することはできません。カレントビューは編集中のビューで、名前が太字で表示されます。

ビュー管理

ビューの作成

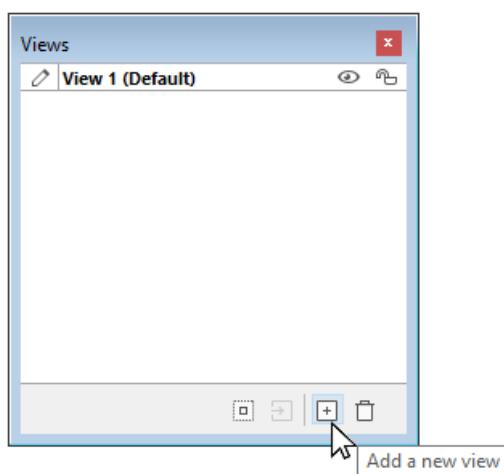
フォーム内で作成されたオブジェクトは、そのフォームの最初のビュー ("View 1") に配置されます。最初のビューは常にデフォルトビューで、名前の後に

(Default) と表示されます。このビューの名前は変更することができます (ビューの名称変更 参照)、デフォルトビューであることに変わりはありません。

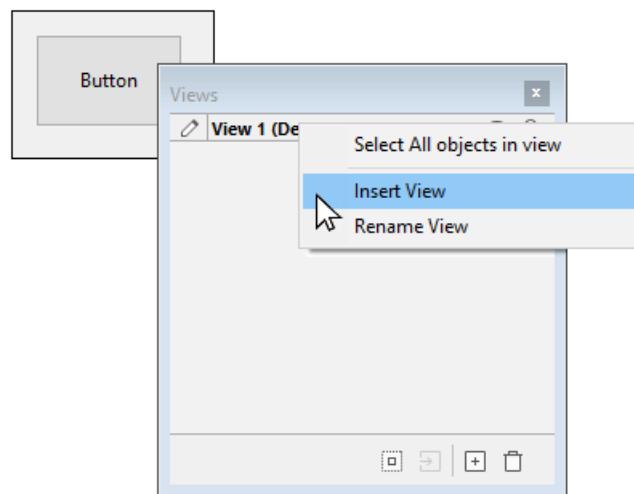


ビューを追加するには 2つの方法があります。

- ビューパレットの下部にある 新しいビューを追加 ボタンをクリックする。



- 既存のビューを右クリックして ビューを追加 を選択する:



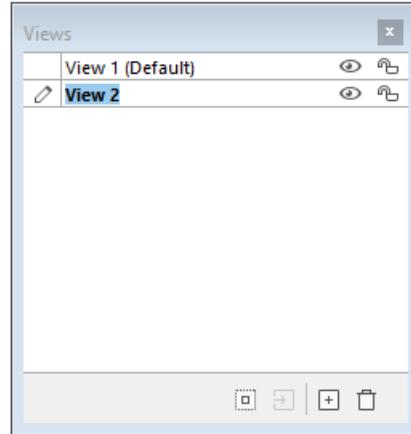
ビューの数に制限はありません。

ビューの名称変更

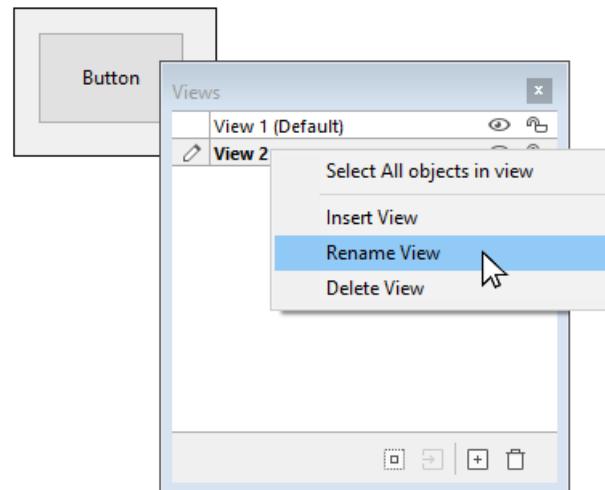
デフォルトでは、ビューの名前は "View" + ビュー番号となっていますが、可読性のためまたは必要性に応じて、これらの名前を変更することができます。

ビューを削除するには、以下のいずれかの方法があります:

- ビューネームを直接ダブルクリックする (ビューが選択されます)。すると、名前が編集可能になります:



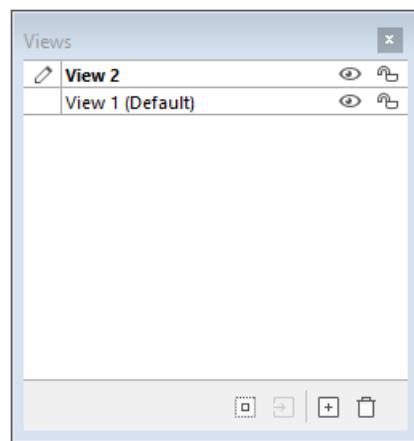
- ビューの名前を右クリックする。すると、名前が編集可能になります:



ビューの並べ替え

ビューパレット内のビューをドラッグ & ドロップすることで、ビューの表示順を変更することができます。

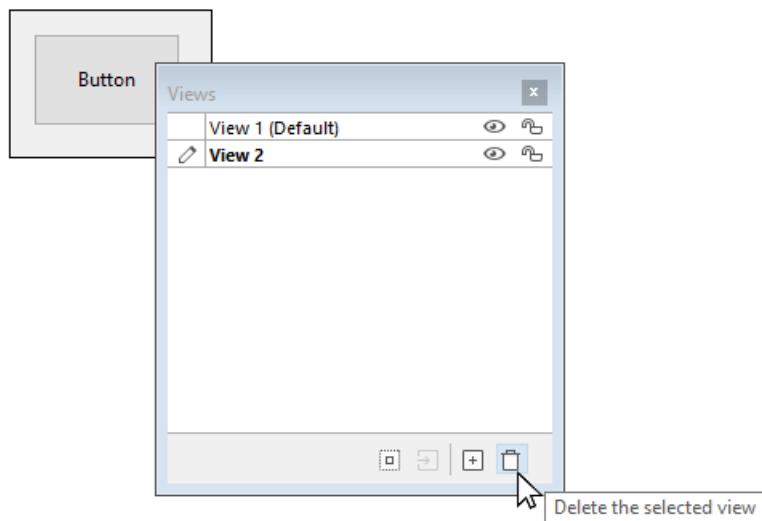
ただし、デフォルトのビューは変更されません:



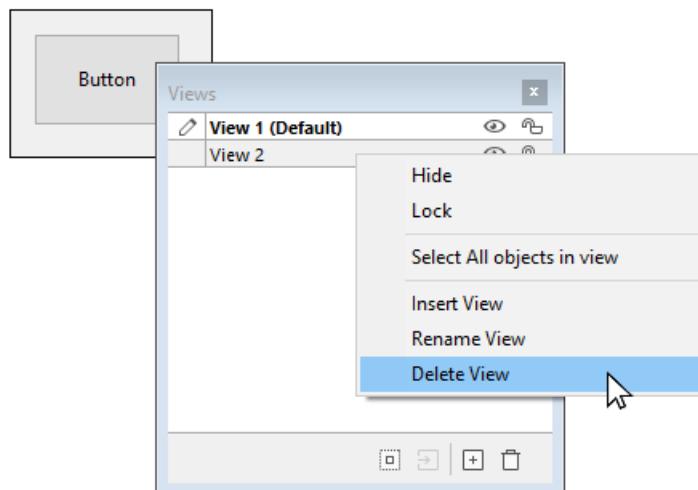
ビューの削除

ビューを削除するには、以下のいずれかの方法があります:

- ビューパレットの下部にある 選択したビューを削除 ボタンをクリックする。



- 既存のビューを右クリックして ビューを削除 を選択する:



ビューを削除すると、その中のオブジェクトは自動的にデフォルトビューに移動します。

ビューを使用する

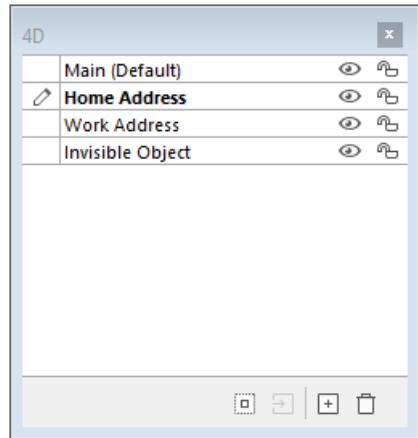
ビューを作成したら、ビューパレットを使用して以下のことがおこなえます:

- ビューにオブジェクトを追加する。
- オブジェクトを他のビューに移動する。
- 同じビュー内の全オブジェクトを 1クリックで選択する。
- ビューごとに表示/非表示を切り替える。
- ビューのオブジェクトをロックする。

ビューにオブジェクトを追加する

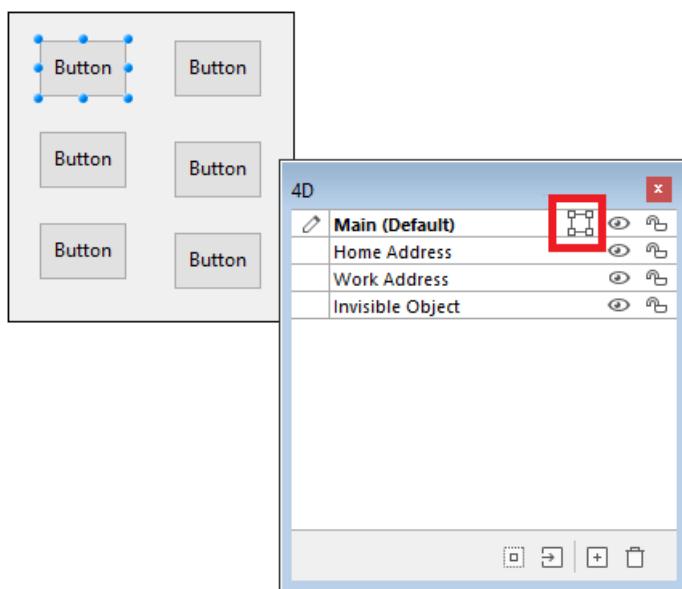
1つのオブジェクトは、1つのビューにのみ属することができます。

他のビューにオブジェクトを作成するには、ビューパレットで目的のビューをクリックし、あらかじめ選択しておきます（カレントビュー に編集アイコンが表示され、名前が太字で表示されます）。



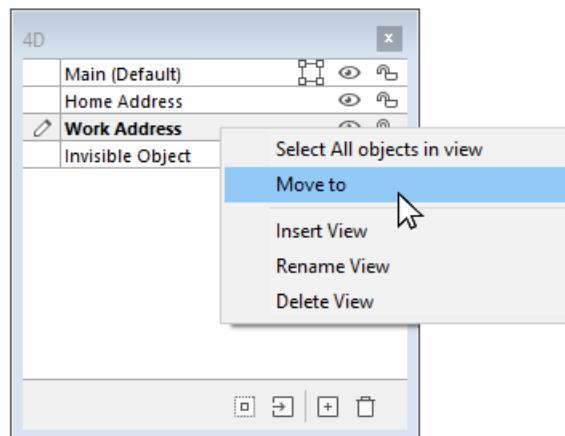
ビュー間のオブジェクト移動

1つ以上のオブジェクトを他のビューに移動することもできます。これを実現するには、ビューを変更したいオブジェクトをフォーム上で選択します。それらのオブジェクトが属するビューが、ビューリスト上に記号で示されます：



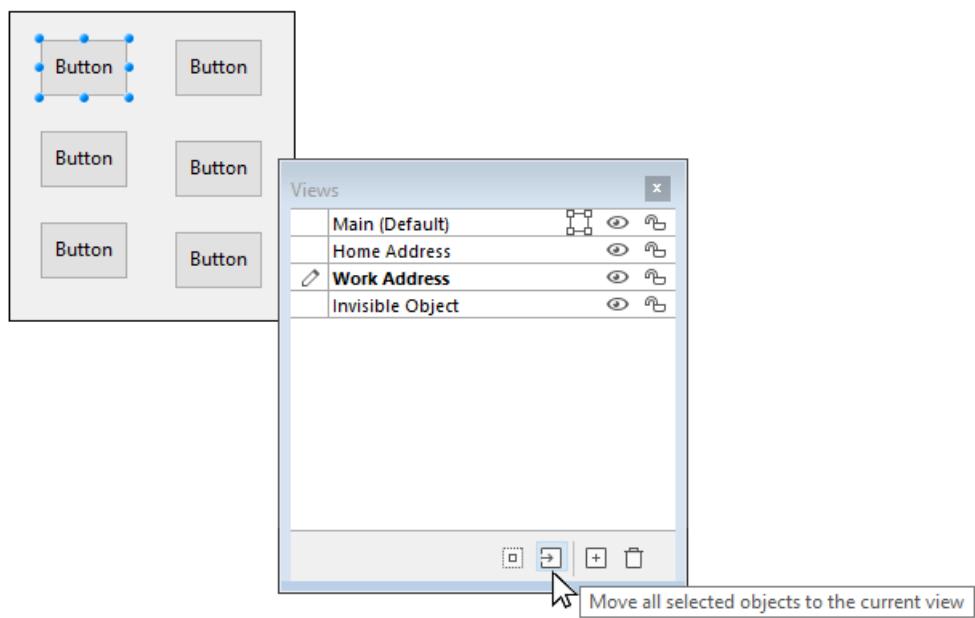
選択したオブジェクトは異なるビューに属していることがあります。

次に移動先のビューを選択し、右クリックして 移行先 を選択します。

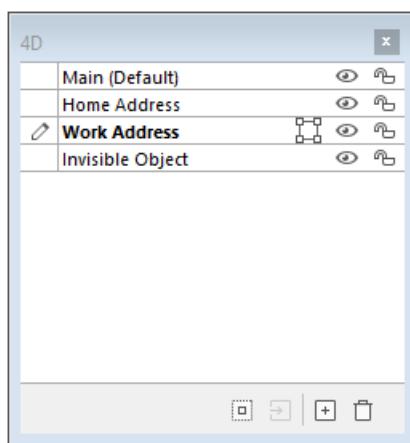


OR

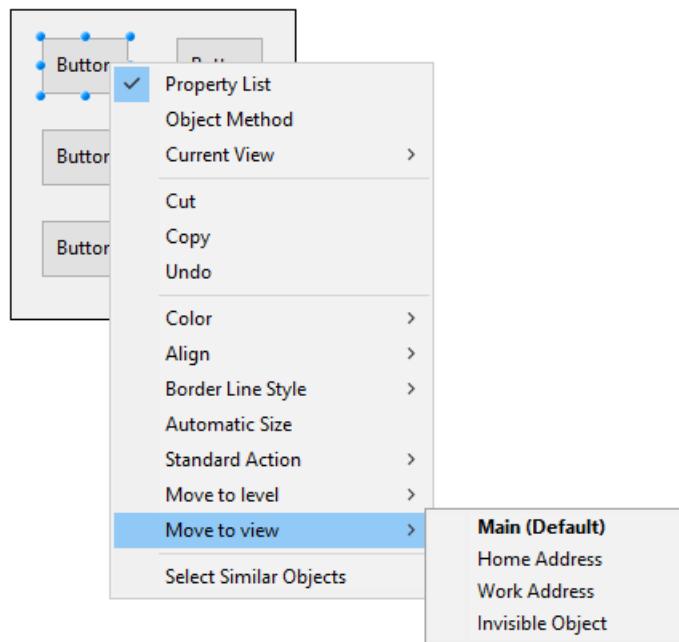
次に移動先のビューを選択し、ビューパレットの下部にある 選択された全てのオブジェクトをカレントのビューに移動 ボタンをクリックします。



選択されているオブジェクトは新しいビューに移動されます:



また、オブジェクトのコンテキストメニューから、オブジェクトを別のビューに移動することもできます。オブジェクトを右クリックして「ビューに移動」を選択し、利用可能なビューのリストから移動先を選択します。

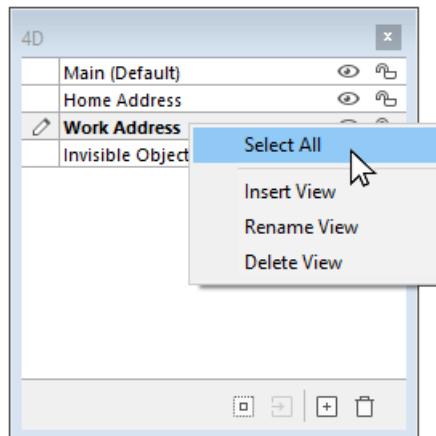


カレントビュー は太字で表示されます。

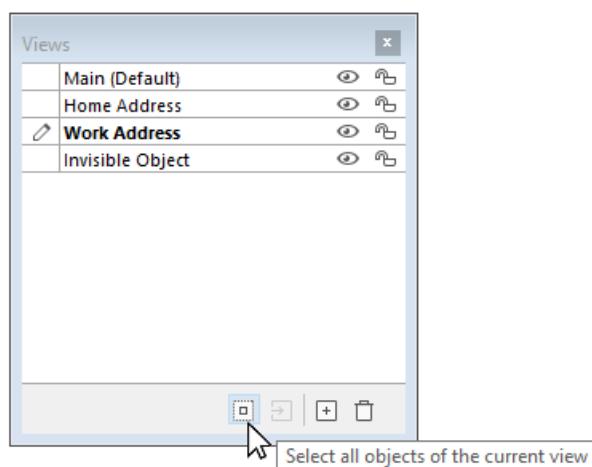
ビューの全オブジェクトを選択する

フォームのカレントページ上で、同じビューに属するすべてのオブジェクトを選択できます。一連のオブジェクトに同じ変更を適用する場合に、この機能は便利です。

これをおこなうにはビュー上で右クリックし、ビュー内の全オブジェクトを選択 をクリックします：



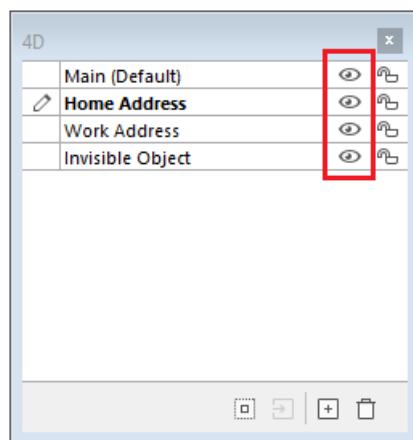
ビューパレットの下部にある カレントビューの全てのオブジェクトを選択 ボタンを使用することもできます。



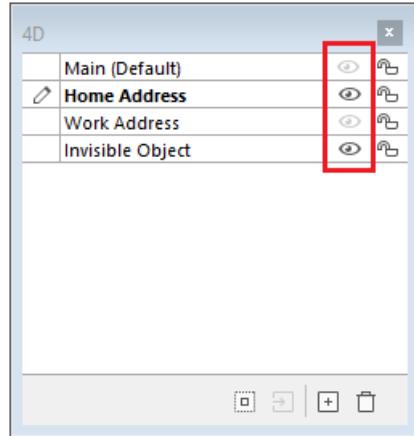
ビューのオブジェクトを表示/非表示にする

フォームのカレントページ上で、ビューに属するオブジェクトの表示/非表示を切り替えることができます。この方法で、フォームの特定オブジェクトに集中して操作することができます。

デフォルトですべてのビューが表示されていて、ビューごとの 表示／非表示 アイコンで示されています：



ビューを隠すにはこのアイコンをクリックします。アイコンがグレーになり、そのビューに属するオブジェクトが非表示となります：



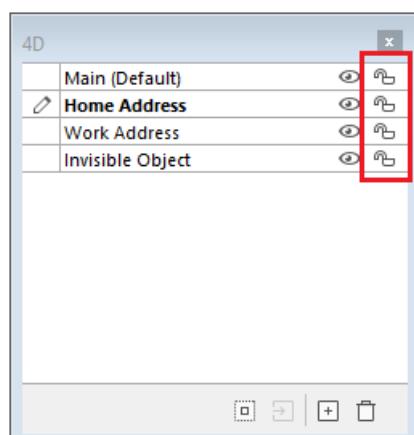
カレントビューは非表示にできません。

非表示のビューを表示にするには、ビューを選択するか、表示/非表示アイコンをクリックします。

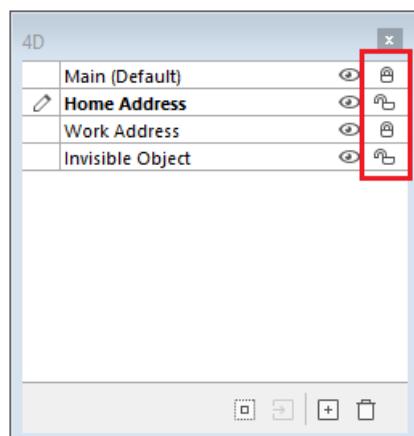
ビューのオブジェクトをロックする

ビューのオブジェクトをロックすることができます。ロックされたオブジェクトは選択・変更・削除できなくなります。また、クリックやマウスドラッグ、同じ種類のオブジェクトを選択コマンドで選択することもできません。この機能は、操作ミスを防ぐのに役立ちます。

デフォルトでは、すべてのビューがロックされていない状態です。これは、ビューごとのロックアイコンで示されています：



ビューをロックするにはこのアイコンをクリックします。南京錠が閉まり、ビューがロックされます：



カレントビューはロックできません。

ビューのロックを解除するには、ビューを選択するか、ロックアイコンをクリックします。

拡大

カレントフォームを拡大/縮小表示することができます。これをおこなうにはツールバーの虫眼鏡をクリックするか、拡大率を直接クリックします。拡大/縮小率は 50%, 100%, 200%, 400% そして 800%です。



- 虫眼鏡ボタンをクリックすると、カーソルも虫眼鏡に変わります。フォーム中をクリックすれば拡大表示され、Shiftキーを押しながらクリックすれば縮小表示されます。
- 拡大率バーをクリックすると、即座に表示が変更されます。

拡大/縮小表示の中もフォームエディターの機能は利用できます (*)。

(*) 技術的な理由から、フォームエディターがズームモードの時にはリストボックスの要素（ヘッダー、カラム、フッター）を選択することはできません。

フォームエディターマクロ

4D のフォームエディターはマクロをサポートしています。1つ以上のアクションを実行するための指示をマクロと呼びます。呼び出されると、マクロは登録された指示を実行し、自動的に指定のアクションをおこないます。

たとえば、定期レポートに特定のフォーマットが指定されている場合（例：テキストによってフォントカラーが赤や緑であるなど）、マクロを作成してフォントカラーの設定を自動でおこなうことができます。4Dフォームエディターのマクロでは、次のことがおこなえます：

- 4Dコードを作成・実行する
- ダイアログを表示する
- オブジェクトを選択する
- フォームやフォームオブジェクトおよびそれらのプロパティを追加・編集・削除する
- プロジェクトファイルを編集する（更新・削除）

フォームエディター用のカスタム機能を定義するため、マクロコードは [クラス関数](#) と [JSON のフォームオブジェクトプロパティ](#) を使用できます。

ホストプロジェクトおよび、プロジェクト内のコンポーネントにてマクロを定義することができます。通常は開発用のコンポーネントにマクロをインストールして使用します。

マクロが呼び出されると、指定されている既存の動作はマクロによってオーバーライドされます。

例題

フォームの左上に "Hello World" アラートボタンを追加するマクロを作成します。

1. プロジェクトの `Sources` フォルダー内に配置された `formMacros.json` ファイルに、次のように書きます：

```
{  
  "macros": {  
    "Add Hello World button": {  
      "class": "AddButton"  
    }  
  }  
}
```

2. 次に、`AddButton` という名前の 4Dクラスを作成します。

3. `AddButton` クラスに次の関数を定義します：

```

Function onInvoke($editor : Object)->$result : Object

var $btnHello : Object

// "Hello" ボタンを作成します
$btnHello:=New object("type"; "button"; \
"text"; "Hello World!"; \
"method"; New object("source"; "ALERT(\"Hello World!\")"); \
"events"; New collection("onClick"); \
"width"; 120; \
"height"; 20; \
"top"; 0; \
"left"; 0)

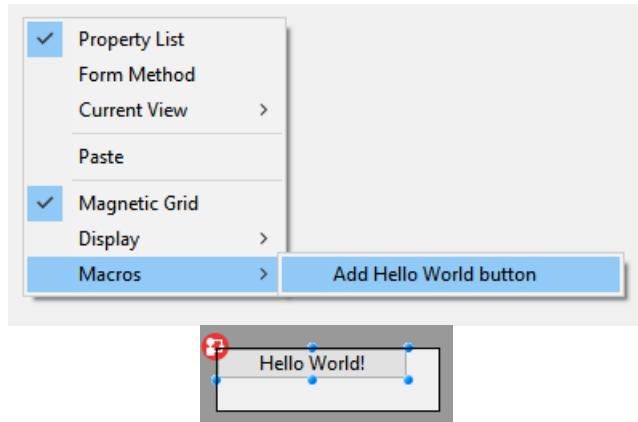
// 現在のページにボタンを追加します
$editor.editor.currentPage.objects.btnHello:=$btnHello

// フォームエディター上で新規作成したボタンを選択します
$editor.editor.currentSelection.clear() //unselect elements
$editor.editor.currentSelection.push("btnHello")

// 4D に変更内容を通知します
$result:=New object("currentSelection"; $editor.editor.currentSelection; \
"currentPage"; $editor.editor.currentPage)

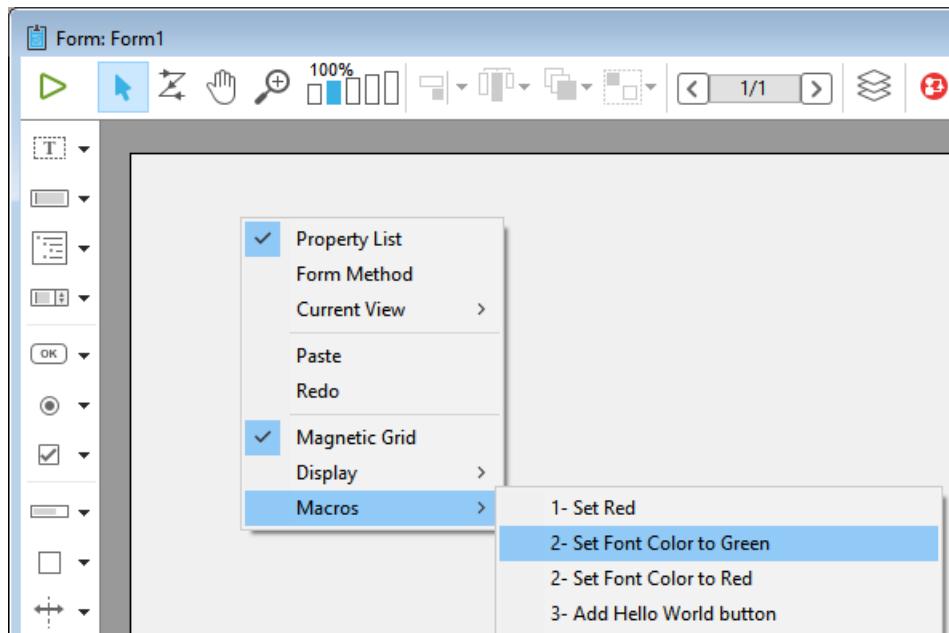
```

マクロを呼び出します:



フォームエディターでマクロを呼び出す

4Dプロジェクトにマクロが定義されていると、フォームエディターのコンテキストメニューを使ってマクロを呼び出すことができます:



このメニューは `formMacros.json` マクロ定義ファイル をもとに作成されています。マクロメニュー項目はABC順に表示されます。

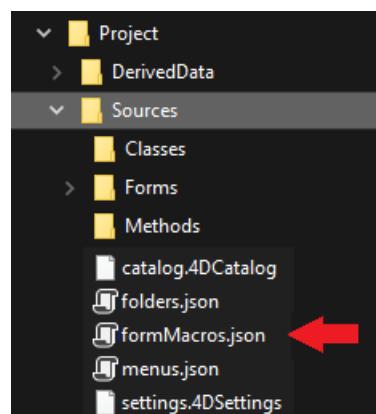
このメニューは、フォームエディター内で右クリックにより開くことができます。選択オブジェクトがある状態や、フォームオブジェクトの上でマクロを呼び出した場合は、それらのオブジェクト名がマクロの `onInvoke` 関数の `$editor.currentSelection` や `$editor.target` パラメーターに受け渡されます。

1つのマクロによって複数の処理を実行することができます。マクロで実行した処理は、フォームエディターの 取消し 機能でも戻すことができます。

マクロファイルの場所

4Dフォームエディターマクロは、プロジェクトあるいはコンポーネントごとに 1つの JSONファイルによって定義されます: `FormMacros.json`。

このファイルは、ホストまたはコンポーネントプロジェクトの Project > Sources フィルラーに配置しなければなりません:



マクロの宣言

`formMacros.json` ファイルの構造は次の通りです:

```
{
  "macros": {
    <macroName>: {
      "class": <className>,
      <customProperty> : <value>
    }
  }
}
```

JSONファイルの説明です:

属性			タイプ	説明
macros			object	定義されたマクロのリスト
	<macroName>		object	マクロ定義
		class	string	マクロクラス名
		<customProperty>	any	(任意) コンストラクターによって取得するカスタム値

カスタムプロパティはマクロの [constructor](#) 関数に受け渡されます。

例題

```
{
  "macros": {
    "Open Macros file": {
      "class": "OpenMacro"
    },
    "Align to Right on Target Object": {
      "class": "AlignOnTarget",
      "myParam": "right"
    },
    "Align to Left on Target Object": {
      "class": "AlignOnTarget",
      "myParam": "left"
    }
  }
}
```

マクロのインスタンス化

プロジェクトおよびコンポーネントにおいてインスタンス化するマクロは、それぞれ [4Dクラス](#) として宣言する必要があります。

クラスの名称は、`formMacros.json` ファイルで `class` 属性に定義した名前と同一でなくてはなりません。

マクロは、アプリケーションの起動時にインスタンス化されます。そのため、関数の追加やパラメーターの編集など、マクロクラスになんらかの変更を加えた場合には、それらを反映するにはアプリケーションを再起動する必要があります。

マクロ関数

マクロクラスは、1つの `Class constructor` のほかに、`onInvoke()` および `onError()` という 2つの関数を持つことができます。

Class constructor

`Class constructor($macro : Object)`

引数	タイプ	説明
\$macro	オブジェクト	<code>formMacros.json</code> ファイルのマクロ宣言オブジェクト

`Class constructor` 関数が定義されている場合、マクロはそれによってインスタンス化されます。

アプリケーションの起動時にクラスがインスタンス化される際に、このコンストラクターが呼び出されます。

[マクロの宣言](#) に追加したカスタムプロパティは、クラスコンストラクターが引数として受け取ります。

例題

`formMacros.json` ファイルに次のようなマクロ宣言をした場合:

```
{
  "macros": {
    "最後に選択したオブジェクトを基準に左揃え": {
      "class": "AlignOnTarget",
      "myParam": "left"
    }
  }
}
```

以下のように書くことができます：

```
// クラス "AlignOnTarget"
Class constructor($macro : Object)
  This.myParameter:=$macro.myParam // left ...
```

onInvoke()

`onInvoke($editor : Object) -> $result : Object`

引数	タイプ	説明
\$editor	オブジェクト	フォームプロパティを格納する Form Editor Macro Proxy オブジェクト
\$result	オブジェクト	マクロによって変更されたフォームプロパティ (任意)

マクロが呼び出されるたびに、`onInvoke` 関数が自動的に実行されます。

呼び出しの際、関数は `$editor.editor` プロパティに、フォームの全要素とそれらの現在値のコピーを受け取ります。つまり、これらのプロパティに対して、任意の処理を実行することができます。

マクロによってオブジェクトを変更・追加・削除した場合、操作を反映させるには最後に結果のプロパティを `$result` に返します。返されたプロパティは解析され、フォームに対して変更が適用されます。戻り値に含まれるプロパティが少ないほど、この処理にかかる時間も削減されます。

`$editor` 引数にて渡されるプロパティは次の通りです：

プロパティ	タイプ	説明
\$editor.editor.form	オブジェクト	フォーム全体
\$editor.editor.file	File	フォームファイルの File オブジェクト
\$editor.editor.name	String	フォームの名称
\$editor.editor.table	number	フォームのテーブル番号。プロジェクトフォームの場合は 0。
\$editor.editor.currentPageNumber	number	現在のページの番号
\$editor.editor.currentPage	オブジェクト	現在のページ (フォームオブジェクトおよび入力順序を格納)
\$editor.editor.currentSelection	コレクション	選択されているオブジェクトの名称のコレクション
\$editor.editor.formProperties	オブジェクト	カレントフォームのプロパティ
\$editor.editor.target	string	マクロ呼び出し時にマウスカーソルが置かれているオブジェクトの名称

マクロによる変更をフォームに反映させたい場合に、`$result` オブジェクトに渡せるプロパティの一覧です。いずれのプロパティも任意です：

プロパティ	タイプ	説明
currentPage	オブジェクト	マクロによって変更されたオブジェクトを含む currentPage
currentSelection	コレクション	マクロによって変更された currentSelection
formProperties	オブジェクト	マクロによって変更された formProperties
editor.groups	オブジェクト	マクロによって変更されたグループ情報
editor.views	オブジェクト	マクロによって変更されたビュー情報
editor.activeView	String	有効なビュー名

たとえば、currentPage と editor.groups の内容が変わった場合には、戻り値を次のように設定します：

```
$result:=New object("currentPage"; $editor.editor.currentPage ; \
"editor"; New object("groups"; $editor.editor.form.editor.groups))
```

method 属性

フォームオブジェクトの `method` 属性を操作する場合、属性値は2通りの方法で定義できます：

- メソッドファイル名あるいはパスを指定する文字列 の使用
- 次の構造を持つオブジェクトの使用：

プロパティ	タイプ	説明
source	文字列	メソッドコード

後者の場合、4D は "objectMethods" フォルダー内に当該オブジェクト名を冠したファイルを作成し、`source` 属性に指定したメソッドコードを格納します。この機能はマクロコードの場合にのみ有効です。

currentPage.objects の \$4dId プロパティ

`$4dId` プロパティは、現在のページにある各オブジェクトについて一意のIDを定義します。このキーは `$result.currentPage` の変更を反映させるのに使用されます：

- フォーム上および `$result` 内のオブジェクトの両方で `$4dId` キーが存在しない場合、そのオブジェクトは作成されます。
- フォーム上で存在する `$4dId` キーが、`$result` 内には存在しない場合、当該オブジェクトは削除されます。
- フォーム上および `$result` 内のオブジェクトの両方で `$4dId` キーが存在する場合、そのオブジェクトは変更されます。

例題

選択されているオブジェクトに対して、フォントカラーを赤に、フォントスタイルをイタリックに変更するマクロ関数を定義します。

```

Function onInvoke($editor : Object)->$result : Object
  var $name : Text

  If ($editor.editor.currentSelection.length>0)
    // 選択されている各オブジェクトの stroke 属性を red に、style 属性を italic に設定します
    For each ($name; $editor.editor.currentSelection)
      $editor.editor.currentPage.objects[$name].stroke:="red"
      $editor.editor.currentPage.objects[$name].fontStyle:="italic"

    End for each

  Else
    ALERT("フォームオブジェクトを選択してください。")
  End if

  // 4Dに変更を通知します
  $result:=New object("currentPage"; $editor.editor.currentPage)

```

onError()

onError(\$editor : Object; \$resultMacro : Object ; \$error : Collection)

引数		タイプ	説明
\$editor		オブジェクト	onInvoke に渡されたオブジェクト
\$resultMacro		オブジェクト	onInvoke によって返されたオブジェクト
\$error		コレクション	エラースタック
	[]errorCode	数値	エラーコード
	[]message	テキスト	エラーの詳細
	[]componentSignature	テキスト	内部コンポーネントのシグネチャー

マクロの実行時にエラーが発生した場合、onError 関数が実行されます。

マクロの実行時に発生したエラーが、マクロの取り消しを不可能にする内容の場合、マクロは実行されません。たとえば次のような場合が該当します：

- 読み取り専用ファイルのスクリプトを変更・削除しようとしたとき
- 同じ内部IDを持つオブジェクトを複数作成しようとしたとき

例題

マクロクラス定義に、次のような汎用的なエラーコードを書くことができます：

```

Function onError($editor : Object; $resultMacro : Object; $error : Collection)
  var $obj : Object
  var $txt : Text
  $txt:=""

  For each ($obj; $error)
    $txt:=$txt+$obj.message+"\n"
  End for each

  ALERT($txt)

```

オブジェクトライブラリ

フォームの作成にあたっては、オブジェクトライブラリを利用することができます。オブジェクトライブラリは、ドラッグ & ドロップやコピー/ペーストするだけでフォームに追加することができる、あらかじめ設定された各種オブジェクトを提供しています。

4D では 2種類のオブジェクトライブラリを利用できます:

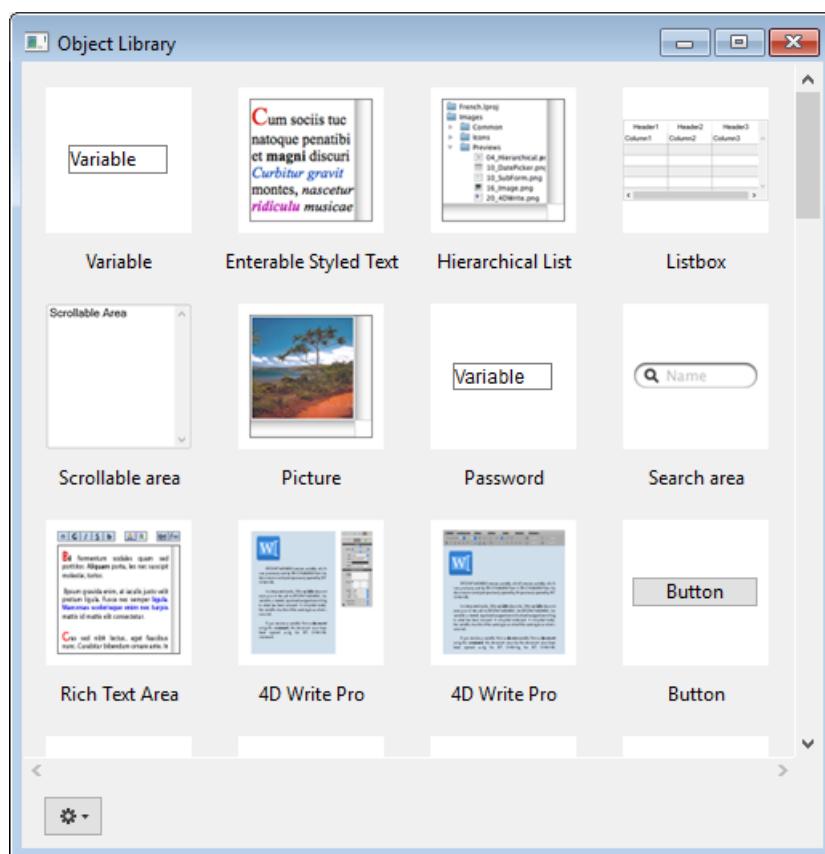
- 標準の設定済みオブジェクトライブラリはすべてのプロジェクトに利用できます
- カスタムオブジェクトライブラリは、開発者自身がお気に入りのフォームオブジェクトや、あるいはプロジェクトフォームそのものをとっておくためのものです

標準のオブジェクトライブラリの使用

標準のオブジェクトライブラリはフォームエディターからアクセスすることができます: ツールバーの右にある次のボタンをクリックします:

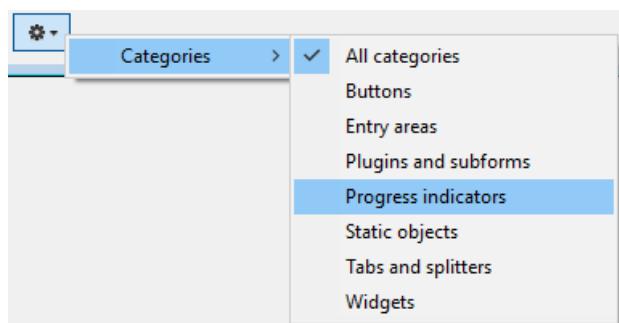


すると、ライブラリが別ウインドウに開きます:



このウインドウには次の主な機能があります:

- Tips 付きプレビューエリア: 中央のエリアには各オブジェクトのプレビューが表示されます。オブジェクトにマウスオーバーすると、オブジェクトに関する情報が Tips として表示されます。



- 表示オブジェクトは カテゴリメニューを使って絞り込むことができます:
- ライブラリのオブジェクトをフォーム上で使うには:
 - オブジェクト上で右クリックし、コンテキストメニューから コピー を選択してフォーム上で同様に ペースト するか、

- ライブラリからオブジェクトをフォーム上にドラッグ & ドロップします。すると、フォームに当該オブジェクトが追加されます。

設定済みオブジェクトライブラリは変更できません。デフォルトオブジェクトを編集したり、設定済みオブジェクトやフォームのライブラリを独自に作るには、カスタムオブジェクトライブラリを作成します（後述参照）。

標準のオブジェクトライブラリにて提供されているオブジェクトについては doc.4d.com で詳しく説明されています。

カスタムオブジェクトライブラリの作成と使用

4Dでカスタムオブジェクトライブラリを作成し、使用することができます。カスタムオブジェクトライブラリとは、任意のオブジェクト（ボタン、テキスト、ピクチャー等）を格納する4Dプロジェクトです。これらのオブジェクトは別のフォームやプロジェクトにて再利用することができます。

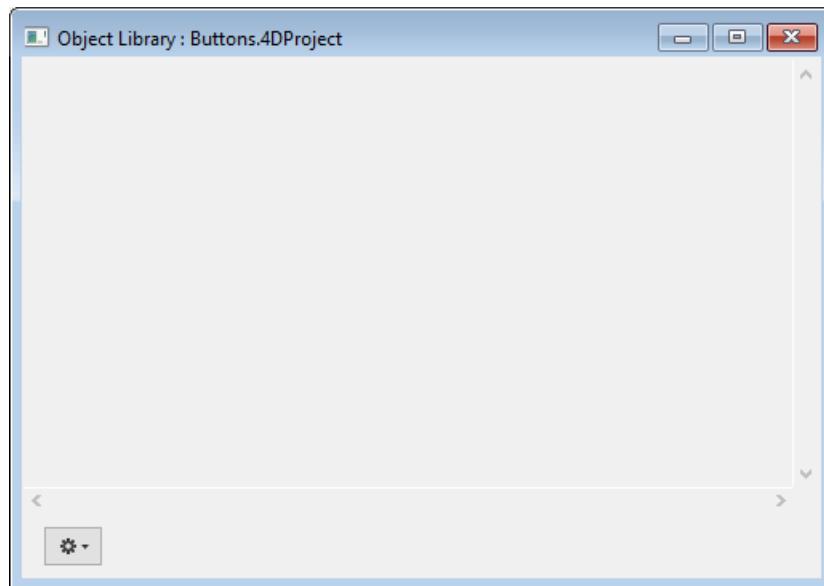
オブジェクトはプロパティおよびオブジェクトメソッドとともに格納されます。ライブラリはドラッグ & ドロップや、コピー/ペースト操作で利用できます。

ライブラリを使用すると、グラフィックファミリーや振る舞いごとにグループ化したフォームオブジェクトを作成できます。

オブジェクトライブラリの作成

オブジェクトライブラリを作成するには、ファイルメニューまたはツールバーから 新規 > オブジェクトライブラリ... を選択します。標準のファイル保存用のダイアログボックスが表示され、オブジェクトライブラリの名前と保存先を指定できます。

ダイアログボックスを受け入れると、4Dはディスク上に新しいオブジェクトライブラリを作成し、ウィンドウに表示します（デフォルトで空です）。



ライブラリは必要なだけ作成できます。macOS上で作成されたライブラリをWindowsで使用すること、あるいはその逆も可能です。

オブジェクトライブラリを開く

一つのオブジェクトライブラリを複数のプロジェクトで同時に開くことはできませんが、一つのプロジェクトで複数のライブラリを開くことは可能です。

カスタムのオブジェクトライブラリを開くには、ファイルメニューまたはツールバーから 開く > オブジェクトライブラリ... コマンドを選択します。標準のファイルを開くダイアログボックスが表示され、オブジェクトライブラリを選択できます。次のファイルタイプが選択できます：

- .4dproject
- .4dz

カスタムオブジェクトライブラリは、実質的には標準の4Dプロジェクトです。プロジェクトをライブラリとして開くと、次のものが公開されます：

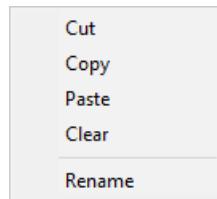
- プロジェクトフォーム
- フォームのページ1

オブジェクトライブラリの構築

ドラッグ & ドロップやコピー/ペースト操作で、オブジェクトをオブジェクトライブラリに配置できます。オブジェクトは、フォームあるいは他のオブジェクトライブラリ（標準のオブジェクトライブラリ含む）からコピーできます。元のオブジェクトとのリンクは保持されないため、オリジナルが編集されてもライブラリのオブジェクトには影響しません。

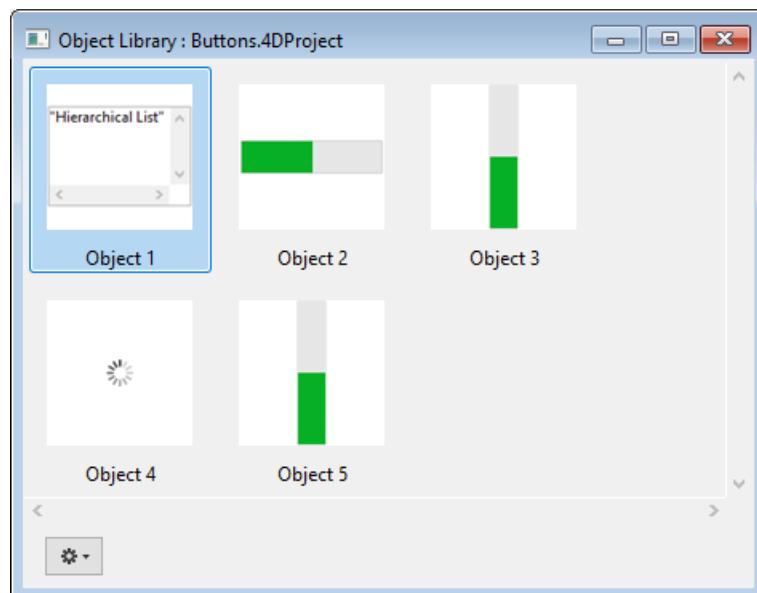
フォームからオブジェクトライブラリにドラッグ & ドロップ操作をおこなうには、4D環境設定のフォームページにて ドラッグ & ドロップを開始 オプションを選択している必要があります。

追加・削除・名称変更などの基本操作はコンテキストメニューあるいはウインドウのオプションメニューから行えます:



- ペーストボードへの カット または コピー
- ペーストボードからオブジェクトを ペースト
- クリア でライブラリからオブジェクトを削除
- 名称変更 でダイアログが開き、オブジェクトの名前を変更することができます。ライブラリ内のオブジェクト名は一意のものでなくではありません。

オブジェクトライブラリには個々のオブジェクト（サブフォーム含む）やオブジェクトグループを格納できます。それぞれのオブジェクトは1つのアイテムとしてグループ化されます:



1つのオブジェクトライブラリには32,000項目まで含めることができます。

オブジェクトは、グラフィックおよび動作に関わるすべてのプロパティとメソッドともにコピーされます。これらのプロパティはオブジェクトがフォームや他のライブラリにコピーされる際にも保持されます。

依存オブジェクト

コピー/ペーストやドラッグ & ドロップで特定のライブラリオブジェクトを使用すると、依存オブジェクトも一緒にコピーされます。たとえば、ボタンをコピーすると、そのオブジェクトに割り当てられていたオブジェクトメソッドもコピーされます。これらの依存オブジェクトはそれのみを直接コピーしたりドラッグ & ドロップしたりすることはできません。

メインのオブジェクトと一緒にライブラリに登録される依存オブジェクトは以下のとおりです:

- リスト
- フォーマット/フィルター
- ピクチャ
- ヘルプTips (フィールドにリンク)
- オブジェクトメソッド

スタイルシート

スタイルシートとは、フォームオブジェクトの属性（テキスト属性など、提供されているほぼすべての属性）の組み合わせをまとめたものです。

アプリケーションのインターフェースを統一するほかにも、スタイルシートの利用には3つの利点があります：

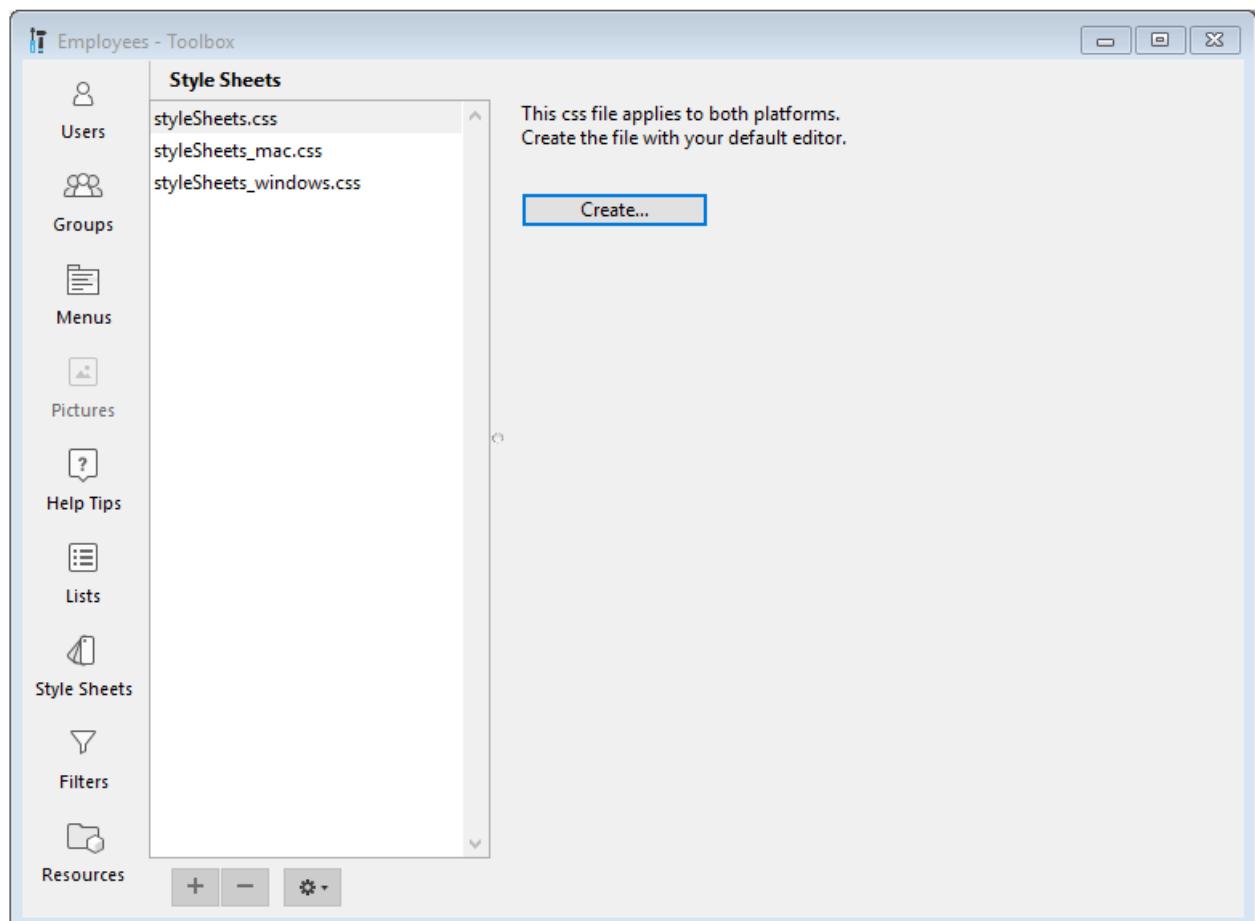
- 開発時間の削減：オブジェクトのさまざまな属性を一括で設定できます。
- メンテナンスの容易化：スタイルシートは、それが使用されているすべてのオブジェクトの外観を一括で設定します。つまり、スタイルシートのフォントサイズを変更するだけで、その変更は同じスタイルシートを使用するすべてのオブジェクトに適用されます。
- マルチプラットフォーム開発のコントロール：macOS と Windows 両用のほか、macOS 専用、Windows 専用のスタイルシートを作成することができます。スタイルシートが設定されていると、4Dは自動で適切なスタイルシートを使用します。

スタイルシートの作成と編集

スタイルシートを作成するには、任意のテキストエディターを使い、".css" 拡張子をファイル名に追加し、プロジェクトの "/SOURCES" フォルダーに保存します。

4D のツールボックスの スタイル ページでは、プラットフォーム専用のスタイルシートを作成・編集するためのショートカットが提供されています。

1. デザインメニューから ツールボックス > スタイルシート を選択するか、ツールバーの ツールボックス アイコンをクリックして スタイル ページを開きます。



2. 作成するスタイルシートを選択し、作成 ボタン（または 編集 ボタン）をクリックします：

Create...

3. 既定のテキストエディターでスタイルシートが開かれます。

スタイルシートファイル

4D は次の3種のスタイルシートファイルを受け付けます：

スタイルシート	プラットフォーム
styleSheets.css	macOS と Windows 両用のデフォルトグローバルスタイルシート
styleSheets_mac.css	macOS 専用の属性スタイル定義用
styleSheets_windows.css	Windows 専用の属性スタイル定義用

これらのファイルはプロジェクトの "/SOURCES" フォルダーに格納されます。また、フォームエディターのツールバーにある、[スタイルシートプレビュー](#) から直接アクセスすることもできます。

スタイルシートアーキテクチャー

4D フォームに適合するよう調整してあるものの、アプリケーションプロジェクトのスタイルシートは基本的に CSS2 のシンタックスと文法に沿っています。

スタイルシートのスタイル規則は二つの部分から構成されています：

- セレクター はスタイルの適用対象を指定します。4D では "オブジェクトタイプ", "オブジェクト名", "クラス", "すべてのオブジェクト", および "属性値" のセレクターが使えます。
- 宣言 は対象に適用すべきスタイルを指定します。複数のまとまった宣言文は宣言ブロックを構成します。CSS 宣言ブロック内の各文はセミコロン ";" で区切り、ブロック全体は中カッコ {} でくくります。

スタイルシートセレクター

オブジェクトタイプ

CSS の 要素セレクターと同様に、スタイルの適用対象をオブジェクトタイプで指定することができます。

まずオブジェクトタイプを指定した後で、中カッコ {} の中に適用するスタイルを宣言します。

オブジェクトタイプとは、フォームオブジェクトの "タイプ" JSON プロパティを指します。

次の例では、button タイプのすべてのオブジェクトについて、表示するフォントを Helvetica Neue に、フォントサイズを 20 ピクセルに指定します：

```
button {
    font-family: Helvetica Neue;
    font-size: 20px;
}
```

複数のオブジェクトタイプに同じスタイルを適用するには、それらのオブジェクトタイプをカンマ "," 区切りで併記し、その後の中カッコ {} 内にスタイルを宣言します：

```
text, input {
    text-align: left;
    stroke: grey;
}
```

オブジェクト名

オブジェクト名はフォーム上で一意的なものため、CSS の ID セレクター と同様に、スタイルの適用対象を指定するのに使えます。

シャープ記号 "#" の後にオブジェクト名を指定し、中カッコ {} の中に適用するスタイルを宣言します。

次の例では、"okButton" というオブジェクト名を持つすべてのオブジェクトについて、表示するフォントを Helvetica Neue に、フォントサイズを 20 ピクセルに指定します：

```
#okButton {
    font-family: Helvetica Neue;
    font-size: 20px;
}
```

クラス

CSS の クラスセレクターと同様に、スタイルの適用対象をフォームオブジェクトの `クラス` 属性で指定することができます。

ドット記号 "." の後にクラス名を指定し、中カッコ {} の中に適用するスタイルを宣言します。

次の例では、`okButtons` クラスを持つすべてのオブジェクトについて、表示するフォントを Helvetica Neue に、フォントサイズを 20 ピクセルに指定します：

```
.okButtons {
    font-family: Helvetica Neue;
    font-size: 20px;
    text-align: center;
}
```

さらに、特定のオブジェクトタイプに限定してスタイルを適用するには、そのオブジェクトタイプの後にドット "." 区切りでクラス名を指定し、その後の中カッコ {} 内にスタイルを宣言します：

```
text.center {
    text-align: center;
    stroke: red;
}
```

4D フォームの JSON式記述においては、フォームオブジェクトにクラス名を設定するには `class` 属性を使います。この属性には一つ以上のクラス名を指定することができます。複数の場合はクラス名を半角スペースで区切ります：

```
class: "okButtons important"
```

すべてのオブジェクト

アスタリスク "*" は、CSS の 全称セレクター と同様に、すべてのフォームオブジェクトを対象にスタイルを適用します。

アスタリスク "*" の後に、中カッコ {} の中に適用するスタイルを宣言します。

次の例では、すべてのオブジェクトの塗りカラーをグレーにします：

```
* {
    fill: gray;
}
```

オブジェクト属性

CSS の 属性セレクターと同様に、フォームオブジェクトの属性に基づいてスタイルの適用対象を指定することができます。

対象とする属性を大カッコ [] 内で指定し、中カッコ {} の中に適用するスタイルを宣言します。

シンタックスの使い方

シンタックス	説明
[attribute]	attribute 属性を持つオブジェクトが対象です
[attribute="value"]	attribute 属性の値が "value" と合致するオブジェクトが対象です
[attribute~="value"]	attribute 属性値 (複数の場合は半角スペース区切り) に "value" と合致するものが含まれているオブジェクトが対象です
[attribute = "value"]	attribute 属性の値が "value" で始まるオブジェクトが対象です

例題

`borderStyle` (境界線スタイル) 属性を持つすべてのオブジェクトの描画色を紫に指定します:

```
[borderStyle]
{
    stroke: purple;
}
```

テキストタイプかつ、タイトルプロパティ (text属性) の値が "Hello" のオブジェクトの文字色を青に指定します:

```
text[text=Hello]
{
    stroke: blue;
}
```

タイトルプロパティ (text属性) の値が "Hello" を含むオブジェクトの描画色を青に指定します:

```
[text~=Hello]
{
    stroke: blue;
}
```

テキストタイプかつ、タイトルプロパティ (text属性) の値が "Hello" で始まるオブジェクトの文字色を黄色に指定します:

```
text[text|=Hello]
{
    stroke: yellow;
}
```

スタイルシート宣言

メディアクエリ

メディアクエリは、アプリケーションにカラースキームを適用するのに利用します。

メディアクエリは、メディア特性と値によって構成されます (例: <media feature>:<value>)。

使用可能なメディア特性:

- `prefers-color-scheme`

使用可能なメディア特性の値:

- light

- ライトモード
- dark
- ダークモード

カラースキームは macOS でのみサポートされています。

例題

ライトモード（デフォルト）およびダークモードにおける、テキストとテキスト背景の色指定を CSS によっておこないます：

```
@media (prefers-color-scheme: light) {  
    .textScheme {  
        fill: LightGrey;  
        stroke: Black;  
    }  
}  
  
@media (prefers-color-scheme: dark) {  
    .textScheme {  
        fill: DarkSlateGray;  
        stroke: LightGrey;  
    }  
}
```

オブジェクト属性

多くのフォームオブジェクト属性をスタイルシートによって指定することができますが、次の属性は除外されます： - `method` - `type` - `class` - `event` - `choiceList`, `excludedList`, `labels`, `list`, `requiredList` (リストタイプ)

フォームオブジェクトの属性は、それらの [JSON名](#)を使って CSS 属性のように指定できます（オブジェクトタイプやメソッド、イベント、リストなどの属性を除く）。

属性マッピング

次の属性については、4D の名称または CSS の名称を使用することができます：

4D	CSS
<code>borderStyle</code>	<code>border-style</code>
<code>fill</code>	<code>background-color</code>
<code>fontFamily</code>	<code>font-family</code>
<code>fontSize</code>	<code>font-size</code>
<code>fontWeight</code>	<code>font-weight</code>
<code>stroke</code>	<code>color</code>
<code>textAlign</code>	<code>text-align</code>
<code>textDecoration</code>	<code>text-decoration</code>
<code>verticalAlign</code>	<code>vertical-align</code>

CSS の属性名を使用する場合、4D に特有の値（例 `sunken`（くぼみ））はサポートされません。

特殊な属性値

- `icon`, `picture`, および `customBackgroundPicture` のように、値として画像のパスを受け付ける属性の場合、次のように書きます:

```
icon: url("/RESOURCES/Images/Buttons/edit.png"); /* 絶対パス */
icon: url("edit.png"); /* フォームファイルを基準とした相対パス */
```

- `fill`, `stroke`, `alternateFill`, `horizontalLineStroke` および `verticalLineStroke` の属性は 3種類のシンタックスを受け付けます:
 - CSS カーネル名: `fill: red;`
 - 16進数カラーコード: `fill: #FF0000;`
 - `rgb()` 関数: `fill:rgb(255,0,0)`
- CSS では禁じられている文字を使用している文字列については、その文字列を单一引用符または二重引用符でくくることができます。たとえば:
 - xliff 参照の場合: `tooltip: ":xcliff:CommonMenuFile";`
 - データソースがフィールド式の場合: `dataSource: "[Table_1:1]ID:1";`

優先順位

4D プロジェクト内でスタイルが競合する場合には、スタイルシートよりもフォームの定義が優先されます。

JSON vs スタイルシート

フォームの JSON式記述とスタイルシートの両方において属性が定義されている場合、4D は JSON ファイルの値を採用します。

これをオーバーライドするには、スタイルシートの値の後に `!important` 宣言を追加します。

例 1:

JSON 式記述	スタイルシート	4D の表示
<code>"text": "Button",</code>	<code>text: Edit;</code>	<code>"Button"</code>

例 2:

JSON 式記述	スタイルシート	4D の表示
<code>"text": "Button",</code>	<code>text: Edit !important;</code>	<code>"Edit"</code>

複数スタイルシート

ランタイムにおいて複数のスタイルシートが存在する場合、それらの優先順位は次のように決まります:

1. 4D フォームはまずデフォルトの CSS ファイル `/SOURCES/styleSheets.css` を読み込みます。
2. 次に、カレントプラットフォーム専用の CSS ファイル `/SOURCES/styleSheets_mac.css` または `/SOURCES/styleSheets_windows.css` がロードされます。
3. その後、JSON フォーム内に CSS ファイルが定義されていれば、それを読み込みます:

- 両プラットフォーム用のファイル:

```
"css": "<path>"
```

- または、両プラットフォーム用に複数のファイル:

```
"css": [
  "<path1>",
  "<path2>"
],
```

- または、プラットフォームごとのファイルリスト:

```
"css": [  
    {"path": "<path>", "media": "mac"},  
    {"path": "<path>", "media": "windows"},  
],
```

ファイルパスは相対パスと絶対パスが使えます。 * 相対パスの基準は JSON フォームファイルです。 * セキュリティのため、絶対パスとして使用できるのはファイルシステムパスに限られます。(例: "/RESOURCES", "/DATA")

参照

[4DフォームとCSS](#) のビデオプレゼンテーションをご覧ください。

ピクチャー

フォーム上で使用されるピクチャーについて、4Dは次のようにサポートしています。

サポートされるネイティブフォーマット

4Dはピクチャーフォーマットのネイティブ管理を統合しています。これは、ピクチャーが変換されることなく、元のフォーマットのまま 4D で格納、表示されることを意味します。(シェイドや透過など) フォーマットにより異なる特定の機能はコピー・ペーストされる際にも保持され、改変なく表示されます。このネイティブサポートは 4D に格納されるすべてのピクチャー (デザインモードでフォームにペーストされた [スタティックピクチャー](#)、ランタイムで [入力オブジェクト](#) にペーストされたピクチャーなど) に対して有効です。

もっとも一般的なフォーマット (例: jpeg、gif、png、tiff、bmp、等) はどちらのフォーマットでもサポートされます。macOS では、PDF フォーマットのエンコーディング/デコーディングも可能です。

サポートされるフォーマットの完全なリストは OS や、マシンにインストールされているカスタムコーデックによって異なります。どのコーデックが利用可能かを調べるためにには、`PICTURE CODEC LIST` コマンドを使用してください。また、データ型の [ピクチャー](#) の項も参照ください。

利用不可能なピクチャーフォーマット

マシン上で利用できないフォーマットのピクチャーに対しては、専用のアイコンが表示されます。アイコンの下部にその拡張子が表示されます。



このアイコンは、そのピクチャーが表示されるべきところに自動的に使用されます:

FirstName :	LastName :	Photo :
Elizabeth	Smith	
Gerry	Mc Namara	
Henry	Portier	

このアイコンは、そのピクチャーがローカルでは表示も編集もできないことを意味します。ですが、中身を改変することなく保存し、他のマシンで表示することは可能です。たとえば、Windows での PDF ピクチャーや、PICT フォーマットのピクチャーなどが該当します。

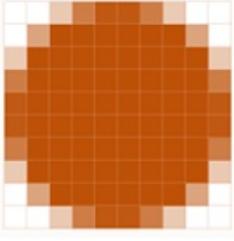
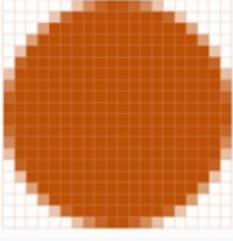
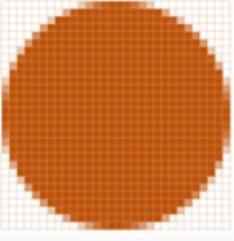
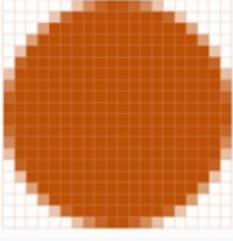
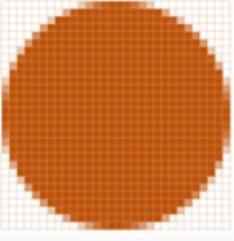
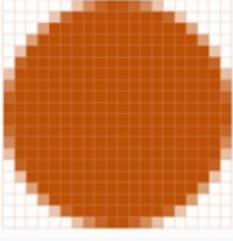
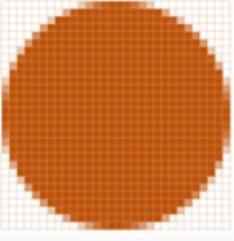
高解像度ピクチャー

4D は macOS および Windows の両方で高解像度ピクチャーの表示をサポートしています。高解像度ピクチャーは、スケール係数または dpi によって定義されます。

スケール係数

従来の標準的なディスプレイと比較して、高解像度ディスプレイは高い画素密度を持ちます。これらの高解像度ディスプレイにおいてピクチャーが正しく表示されるには、適用するスケール係数 (例: 2倍、3倍など) に応じてその画素数を増やす必要があります。

高解像度のピクチャーを使う場合、ピクチャーネームに "@nx" (n: スケール係数) を付けてスケール係数を指定することができます。下のテーブルの例でも、高解像度ピクチャーの名前に、`circle@2x.png`, `circle@3x.png` といった形でスケール係数が指定されています。

表示タイプ	スケール係数	例題						
標準解像度	1:1 ピクセル密度	<p>1x</p>  <p>circle.png</p>						
高解像度	ピクセル密度は2、または3の係数で増加	<table border="1"> <tr> <td>2x</td> <td>3x</td> </tr> <tr> <td></td> <td></td> </tr> <tr> <td>circle@2x.png</td> <td>circle@3x.png</td> </tr> </table>	2x	3x			circle@2x.png	circle@3x.png
2x	3x							
								
circle@2x.png	circle@3x.png							

"@nx" で定義された高解像度ピクチャーは、次のオブジェクトで使用できます。

- [スタティックピクチャー](#)
- [ボタン/ラジオボタン/チェックボックス](#)
- [ピクチャーボタン/ピクチャーポップアップメニュー](#)
- [タブコントロール](#)
- [リストボックスヘッダー](#)
- [メニューアイコン](#)

4D は自動的に最高解像度のピクチャーを優先します。4D は自動的に最高解像度のピクチャーを優先します。コマンドまたはプロパティが *circle.png* を指定していたとしても、*circle@3x.png* があれば、それを使用します。

解像度の優先順位付けはスクリーン上のピクチャー表示にのみ適用され、印刷に関しては自動適用されないことに留意が必要です。

DPI

高解像度が自動的に優先されますが、スクリーンやピクチャーの dpi (*)、およびピクチャー形式によって、動作に違いが生じることがあります：

演算子	動作
ドロップ、ペースト	<p>ピクチャーの設定：</p> <ul style="list-style-type: none"> ● 72dpi または 96dpi - ピクチャーは 中央合わせ 表示され、ピクチャーを表示しているオブジェクトは同じピクセル数です。 ● その他の dpi - ピクチャーは "スケーリング" 表示され、ピクチャーを表示しているオブジェクトのピクセル数は (ピクチャーのピクセル数 / ピクチャーの dpi) * (スクリーンの dpi) です。 ● dpi なし - ピクチャーは "スケーリング" 表示されます。
自動サイズ (フォームエディターのコンテキストメニュー)	<p>ピクチャーの表示が：</p> <ul style="list-style-type: none"> ● スケーリング - ピクチャーを表示しているオブジェクトのピクセル数は (ピクチャーのピクセル数 / ピクチャーの dpi) * (スクリーンの dpi) にリサイズされます。 ● スケーリング以外 - ピクチャーを表示しているオブジェクトは、ピクチャーと同じピクセル数です。

(*) 通常は macOS = 72dpi, Windows = 96dpi

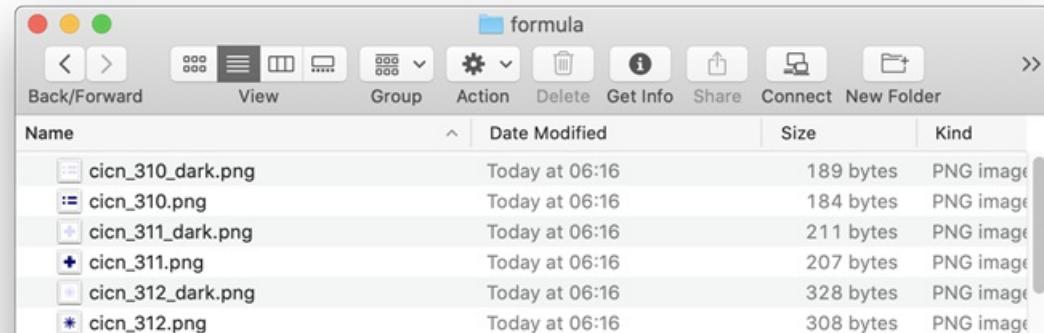
ダークモード (macOS のみ)

フォームがダーススキームを使用 している場合に、標準のピクチャーの代わりに使用する専用のピクチャーやアイコンを定義することができます。

ダークモードピクチャーは以下のように定義します:

- ダークモードピクチャーは、標準（ライトスキーム）バージョンと同じ名前で、"_dark"という接尾辞が付きます。
- ダークモードピクチャーは、標準バージョンの隣に保存します。

ランタイム時に、4D は [現在のフォームのカラースキーム](#) に応じて、ライト用またはダーク用のピクチャーを自動的にロードします。



ピクチャー上のマウス座標

4D では、[ピクチャー式](#) が設定された [入力オブジェクト](#) をクリック、またはホバーした際のマウスのローカル座標を取得できます。これはスクロールやズーム処理がおこなわれている場合でも可能です。このピクチャーマップに似た機構は、たとえば地図作製ソフトウェアのインターフェースや、スクロール可能なボタンバーを管理するのに使用できます。

座標は `MouseX` と `MouseY` [システム変数](#) に返されます。座標はピクセル単位で表現され、ピクチャーの左上隅が起点 (0,0) となります。マウスがピクチャの座標の外側にある場合には、`MouseX` と `MouseY` には -1 が返されます。

これらの値は、[On Clicked](#)、[On Double Clicked](#)、[On Mouse up](#)、[On Mouse Enter](#)、あるいは [On Mouse Move](#) フォームイベントの一部として取得することができます。

JSON プロパティリスト

この一覧は、すべてのフォームプロパティを JSON名順に並べてまとめたものです。プロパティ名をクリックすると、詳細説明に移動します。

"フォームプロパティ" の章では、プロパティリストにおけるテーマと名称で各プロパティを紹介しています。

a - c - d - e - f - h - i - m - p - r - s - w

プロパティ	説明	とりうる値
a		
<code>bottomMargin</code>	垂直マージン値 (ピクセル単位)	最小値: 0
c		
<code>colorScheme</code>	フォームのカラースキーム	"dark", "light"
d		
<code>destination</code>	フォームタイプ	"detailScreen", "listScreen", "detailPrinter", "listPrinter"
e		
<code>entryOrder</code>	入力フォームで Tab キーや 改行キーが使用されたときに、アクティブオブジェクトが選択される順番。	4Dフォームオブジェクト名のコレクション
<code>events</code>	オブジェクトまたはフォームについて選択されているイベントのリスト	イベント名のコレクション。例: ["onClick", "onDataChange"...].
f		
<code>formSizeAnchor</code>	フォームサイズを定義するために使用するオブジェクトの名前 (最小長さ: 1)	4Dオブジェクトの名前
h		
<code>height</code>	フォームの高さ	最小値: 0
i		
<code>inheritedForm</code>	継承フォームを指定します	テーブルまたはプロジェクトフォームの名前 (文字列), フォームを定義する .json ファイルへの POSIX パス (文字列), またはフォームを定義するオブジェクト
<code>inheritedFormTable</code>	継承フォームがテーブルに属していれば、そのテーブル	テーブル名またはテーブル番号
m		
<code>markerBody</code>	詳細マーカーの位置	最小値: 0
<code>markerBreak</code>	ブレークマーカーの位置	最小値: 0
<code>markerFooter</code>	フッターマーカーの位置	最小値: 0
<code>markerHeader</code>	ヘッダーマーカーの位置	最小値 (整数): 0; 最小値 (整数配列): 0
<code>memorizeGeometry</code>	フォームウインドウが閉じられた時に、フォームパラメーターを記憶	true, false
<code>menuBar</code>	フォームと関連づけるメニューバー	有効なメニューバーの名称
<code>method</code>	プロジェクトメソッド名	存在するプロジェクトメソッドの名前
p		

<code>pages</code>	ページのコレクション（各ページはオブジェクトです）	ページオブジェクト どりうる値
<code>pageFormat</code>	object	利用可能な印刷プロパティ
<code>r</code>		
<code>rightMargin</code>	水平マージン値 (ピクセル単位)	最小値: 0
<code>s</code>		
<code>shared</code>	フォームがサブフォームとして利用可能かを指定します	true, false
<code>w</code>		
<code>width</code>	フォームの幅	最小値: 0
<code>windowMaxHeight</code>	フォームウインドウの最大高さ	最小値: 0
<code>windowMaxWidth</code>	フォームウインドウの最大幅	最小値: 0
<code>windowMinHeight</code>	フォームウインドウの最小高さ	最小値: 0
<code>windowMinWidth</code>	フォームウインドウの最小幅	最小値: 0
<code>windowSizingX</code>	フォームウインドウの高さ固定	"fixed", "variable"
<code>windowSizingY</code>	フォームウインドウの幅固定	"fixed", "variable"
<code>windowTitle</code>	フォームウインドウのタイトルを指定します	フォームウインドウの名前。

アクション

メソッド

フォームに関連づけられたメソッドへの参照。フォームメソッドを使用してデータとオブジェクトを管理することができます。ただし、これら目的には、オブジェクトメソッドを使用する方が通常は簡単であり、より効果的です。[特化されたメソッド](#) 参照。

フォームメソッドは呼び出す必要がありません。メソッドが関連づけられているフォームに関わるイベントが発生した場合、4D は自動的にフォームメソッドを呼び出します。

メソッド参照にはいくつかのタイプが利用可能です:

- 標準のプロジェクトメソッドファイルパス:

`method.4dm`

このタイプの参照は、当該メソッドファイルがデフォルトの場所 ("sources/{TableForms/numTable} | {Forms}/formName/") にあることを示します。この場合、エディター上でフォームメソッドに対して操作（名称変更、複製、コピー/ペーストなど）がおこなわれると、4D はこれらの変更を自動的にフォームメソッドに反映させます。

- 拡張子を省いた既存のプロジェクトメソッド名: `myMethod`。この場合、フォームエディターで操作がおこなわれても、4D はそれらの変更を自動反映しません。

- .4dm 拡張子を含むカスタムのメソッドファイルパス:

`MyMethods/myFormMethod.4dm`。ファイルシステムも使用できます:

`/RESOURCES/Forms/FormMethod.4dm`。この場合、フォームエディターで操作がおこなわれても、4D はそれらの変更を自動反映しません。

JSON 文法

名称	データタイプ	とりうる値
method	テキスト	フォームメソッドの標準またはカスタムのファイルパス、またはプロジェクトメソッド名

フォームプロパティ

カラースキーム

配色プロパティは、macOS でのみ適用されます。

このプロパティは、フォームのカラースキームを定義します。このプロパティは、フォームのカラースキームを定義します。これは、フォームに対して以下の 2つのオプションのいずれかに変更することができます：

- dark - 暗い背景に明るいテキスト
- light - 明るい背景に暗いテキスト

定義されたカラースキームを CSS で上書きすることはできません。

JSON 文法

名称	データタイプ	とりうる値
colorScheme	string	"dark", "light"

Pages

各フォームは、少なくとも 2つのページで構成されています：

- ページ0 (背景ページ)
- ページ1 (メインページ)

詳細については [フォームのページ](#) を参照ください。

JSON 文法

名称	データタイプ	とりうる値
pages	collection	ページのコレクション (各ページはオブジェクトで、ページ0 は最初の要素です)

フォーム名

このプロパティはフォームそのものの名称で、4Dランゲージで名前によってフォームを参照するのに使用されます。フォーム名は、4Dの [識別子の命名規則](#) に準じたものでなければなりません。

JSON 文法

フォーム名は、form.4Dform ファイルを格納するフォルダーの名前で定義されます。詳しくは [プロジェクトのアーキテクチャー](#) を参照ください。

フォームタイプ

フォームのタイプ、つまり その出力先によって、当該フォームで利用できる機能が定義されます。たとえば、[マーカー](#) はリスト (出力) テーブルフォームでのみ設定できます。

データベースの各テーブルは通常、少なくとも 2つのテーブルフォームを持ちます。1つは画面上にレコードを一覧表示するためのもので、もう 1つはレコードを 1件ずつ表示するためのものです (データ入力や修正に使用):

- 出力フォーム - 出力フォーム (または リストフォーム) は、レコードのリストを、1レコードにつき 1行で表示します。クエリの結果は出力フォームに表示され、ユーザーが行をダブルクリックすると、そのレコード用に入力フォームが表示されます。

ID :	name :
1	Friends
3	Work
4	Personal
5	Family

- 入力フォーム - データ入力に使用されます。1つの画面に 1件のレコードが表示され、一般的には、レコードの編集を保存・キャンセルするためのボタンや、レコード間を移動するためのボタン (先頭レコード、最終レコード、前レコード、次レコード等) を備えています。

Category		1 / 4
ID :	136	
name :		

サポートされるタイプは、フォームのカテゴリーによって異なります:

フォームタイプ	JSON 文法	説明	サポートされているフォーム
詳細フォーム	detailScreen	データ入力・修正用の表示フォーム	プロジェクトフォームとテーブルフォーム
印刷用詳細フォーム	detailPrinter	1ページにつき 1レコードの印刷レポート (請求書など)	プロジェクトフォームとテーブルフォーム
リストフォーム	listScreen	レコードを画面上に一覧表示するフォーム	テーブルフォーム
印刷用リストフォーム	listPrinter	レコード一覧の印刷レポート	テーブルフォーム
なし	<i>no destination</i>	特定の機能を持たないフォーム	プロジェクトフォームとテーブルフォーム

JSON 文法

名称	データタイプ	とりうる値
destination	string	"detailScreen", "listScreen", "detailPrinter", "listPrinter"

継承されたフォーム名

このプロパティは、現在のフォームに [継承するフォーム](#) を指定します。

テーブルフォームを継承する場合は、[継承されたフォームテーブル](#) プロパティにテーブルを設定します。

継承を解除するには、プロパティリストで <なし> を選択します (JSON では " ")。

JSON 文法

名称	データタイプ	とりうる値
inheritedForm	string	テーブルまたはプロジェクトフォームの名前、フォームを定義する .json ファイルへの POSIXパス、またはフォームを定義するオブジェクト

継承されたフォームテーブル

このプロパティは、現在のフォームに [継承するテーブルフォーム](#) が属するデータベーステーブルを指定します。

プロジェクトフォームを継承する場合は、プロパティリストで <なし> を選択します (JSON では " ")。

JSON 文法

名称	データタイプ	とりうる値
inheritedFormTable	string または number	テーブル名またはテーブル番号

サブフォームとして公開

コンポーネントフォームをホストアプリケーションの [サブフォーム](#) として選択するには、明示的に共有されている必要があります。このプロパティが選択されると、フォームがホストアプリケーションで公開されます。

公開されたサブフォームとして指定できるのは、プロジェクトフォームのみです。

JSON 文法

名称	データタイプ	とりうる値
shared	boolean	true, false

配置を記憶

このオプションがチェックされていると、Open form window コマンドに * 演算子を渡して開かれたウィンドウが閉じられるとき、そのフォームの特定のプロパティ値については、それらがセッション中に変更された場合、4D によって自動的に保存されます：

- カレントページ
- それぞれのフォームオブジェクトの配置・大きさ・表示状態 (リストボックス列のサイズと表示状態も含む)。

このオプションは、OBJECT DUPLICATE コマンドを使用して作成されたオブジェクトに対しては無効です。このコマンドを使用したときに使用環境を復元させるには、開発者がオブジェクトの作成・定義・配置の手順を再現しなければなりません。

このオプションが選択されているとき、一部のオブジェクトに置いては [値を記憶](#) のオプションが選択可能になります。

JSON 文法

名称	データタイプ	とりうる値
memorizeGeometry	boolean	true, false

参照

[値を記憶](#)

ウィンドウタイトル

ウィンドウタイトルは、アプリケーションモードで `Open window` や `Open form window` コマンドを用いてフォームを開く際に使用されます。ウィンドウタイトルはウィンドウのタイトルバーに表示されます。

動的参照を使用して、フォームのウィンドウタイトルを定義することもできます:

- Resourcesフォルダーに保存された、標準の XLIFF 参照
- テーブル/フィールドラベル: 適用できるシンタックスは `<?[TableName]FieldNum>` または `<?[TableNum]FieldNum>` です。
- 変数またはフィールド: 適用できるシンタックスは `<[VariableName]>` または `<[TableName]FieldName>`。フィールドや変数の現在の値が ウィンドウタイトルとして表示されます。

ウィンドウタイトルの最大文字数は 31 です。

JSON 文法

名称	データタイプ	とりうる値
windowTitle	string	テキストまたは参照としてのウィンドウ名

フォームサイズ

4Dでは、フォームと [ウィンドウ](#) の両方のサイズを設定することができます。これらのプロパティは相互に依存しており、アプリケーションのインターフェースはこれらの相互作用によってもたらされます。

サイズオプションは、サイズを決めるもの オプションの値に依存します。

サイズを決めるもの

- **自動サイズ:** フォームサイズは、すべてのオブジェクトを表示するために必要なサイズと、[水平マージン](#) および [垂直マージン](#) フィールドへ入力されたマージン値 (ピクセル単位) を合計したものになります。

自動サイズのウィンドウを用いて、オフスクリーンエリア (ウィンドウの矩形境界線の外側のエリア) に配置したアクティブオブジェクトを使用したい場合にこのオプションを選択することができます。このオプションを選択すると、これらのオブジェクトによりウィンドウサイズが変更されなくなります。

- **サイズを設定:** フォームサイズは [幅](#) および [高さ](#) フィールドに入力された値 (ピクセル単位) により決まります。
- **オブジェクト名:** フォームサイズは、選択したフォームオブジェクトの位置により決まります。たとえば、表示されるエリアの右下部分に置かれているオブジェクトを選択した場合は、左上端が起点であり、右下端が選択したオブジェクトの右下端となる矩形にマージン値を加算したものがフォームサイズになります。

出力フォームの場合は [水平マージン](#) または [幅](#) フィールドだけが利用可能です。

JSON 文法

名称	データタイプ	とりうる値
formSizeAnchor	string	フォームサイズを定義するために使用するオブジェクトの名前

高さ

[フォームサイズ](#) が サイズを設定 の場合のフォームの高さ (ピクセル単位) です。

JSON 文法

名称	データタイプ	とりうる値
height	number	整数値

水平マージン

[フォームサイズ](#) が 自動サイズ または オブジェクト名 の場合に、フォームの右マージンに追加する値 (ピクセル単位) です。

この値は、ラベルエディターで使用されるフォームの右マージンも決定します。

JSON 文法

名称	データタイプ	とりうる値
rightMargin	number	整数値

垂直 マージン

[フォームサイズ](#)が「自動サイズ」または「オブジェクト名」の場合に、フォームの下マージンに追加する値（ピクセル単位）です。

この値は、ラベルエディターで使用されるフォームの上マージンも決定します。

JSON 文法

名称	データタイプ	とりうる値
bottomMargin	number	整数値

幅

[フォームサイズ](#)が「サイズを設定」の場合のフォームの幅（ピクセル単位）です。

JSON 文法

名称	データタイプ	とりうる値
width	number	整数値

マーカー

これらのプロパティを使用して、テーブルフォーム の縦ルーラー上でマーカーの位置を精密に指定することができます。マーカーは主に出力フォームで使用されます。マーカーはリストされる情報を制御し、ヘッダーやブレーク、詳細、フッターを設定します。各エリア内に配置されたオブジェクトは、対応する場所に表示や印刷されます。

フォームが出力フォームとして使用されるときは、画面用であれ印刷用であれ、マーカー設定が考慮されて各エリアが指定された場所に表示/印刷されます。また、サブフォームエリア内のリストフォームとしてフォームが使用される場合も出力マーカーは有効です。ただし、フォームが入力フォームとして使用される場合、マーカー設定は無視されます。

これらのエリアに配置されたオブジェクトに割り当てられたメソッドは、適切なイベントが有効にされていれば、エリアが表示/印刷されるときに実行されます。たとえば、ヘッダーエリアに配置されたオブジェクトのメソッドは `On Header` イベントで実行されます。

フォームブレーク

フォームブレークエリアはレコードリストの下に一回だけ表示あるいは印刷されます。

ブレークエリアは詳細マーカー (D) とブレークマーカー (B0) の間です。レポートに [複数のブレークエリア](#)を追加することもできます。

ブレークエリアの大きさは変更することができます。レコードデータではない情報（説明や日付、時刻など）のほか、線やその他のグラフィック要素を表示するためにブレークエリアを使用できます。印刷レポートでは、小計などのサマリ計算をおこなって印刷するために使用できます。

JSON 文法

名称	データタイプ	とりうる値
markerBreak	integer integer collection	ブレークマーカーの位置（ピクセル単位）、またはブレークマーカー位置コレクションを指定します。 最小値: 0

フォーム詳細

フォーム詳細エリアは、レポート中のレコード毎に画面に表示されたり印刷されたりします。詳細エリアはヘッダーマーカー (H) と詳細マーカー (D) の間です。

詳細エリアの大きさは変更することができます。詳細エリアに配置したオブジェクトはレコード毎に表示または印刷されます。主にフィールドや変数を配置して、各レコードの情報を表示/印刷しますが、他のオブジェクトを配置することもできます。

JSON 文法

名称	データタイプ	とりうる値
markerBody	integer	詳細マーカーの位置。最小値: 0

フォームフッター

フォームフッターエリアは、画面ではレコードリストの下の表示されます。印刷レポートの場合は、各ページの一番下に印刷されます。フッターエリアはブレークマーカー (B0) とフッターマーカー (F) の間です。フッターエリアの大きさは変更することができます。

フッターエリアを使用して画像、ページ番号、日付、また各ページの下部に配置したいテキストを何でも印刷できます。通常、画面表示用の出力フォームでは、フッターエリアに検索や並べ替え、印刷などをおこなうためのボタンを配置します。すべてのアクティブオブジェクトを配置できます。

JSON 文法

名称	データタイプ	とりうる値
markerFooter	integer	最小値: 0

フォームヘッダー

フォームヘッダーエリアは画面の一番上に表示され、また印刷時には各ページの一番上に印刷されます。ヘッダーエリアはフォームの一番上からヘッダーマーカー (H) までの間です。ヘッダーエリアの大きさは変更することができます。ヘッダーエリアには列のタイトル、フォームの説明、その他の情報、会社ロゴなどの画像を配置します。

サブフォームとして表示される出力フォーム、あるいは `DISPLAY SELECTION` や `MODIFY SELECTION` コマンドを使用して表示される出力フォームのヘッダーエリアにアクティボオブジェクトを配置して使用することもできます。以下のようなアクティボオブジェクトを配置できます:

- ボタン、ピクチャーボタン
- コンボボックス、ドロップダウンリスト、ピクチャーポップアップメニュー
- 階層リスト、リストボックス
- ラジオボタン、チェックボックス、3Dチェックボックス
- 進捗インジケーター、ルーラー、ステッパー、スピナー

`addSubrecord` (サブレコード追加) や `cancel`、`automaticSplitter` (自動スプリッター) などの標準アクションをボタンに割り当てるることができます。ヘッダーエリアに配置したアクティボオブジェクトでは以下のイベントを使用できます: `On Load`, `On Clicked`, `On Header`, `On Printing Footer`, `On Double Clicked`, `On Drop`, `On Drag Over`, `On Unload`。フォームメソッドが `On Header` イベントで呼び出されるのは、エリアのオブジェクトメソッドが呼び出された後になります。

フォームには、[追加のヘッダーエリア](#) を作成し、追加ブレークと関連づけることができます。レベル1ヘッダーは、最初のソートフィールドによりグループ化されたレコードが印刷される直前に印刷されます。

JSON 文法

名称	データタイプ	とりうる値
markerHeader	integer integer collection	ヘッダーマーカーの位置 (ピクセル単位)、またはヘッダーマーカー位置のコレクションを指定します。 最小値: 0

追加マーカーの作成

レポート用に追加のブレークエリアやヘッダーエリアを作成できます。これらの追加されたエリアを使用して、小計などの計算結果をレポートに印刷したり、その他の情報を効果的に表示することができます。

追加工業は、[フォームブレーク](#) と [フォームヘッダー](#) のプロパティにマーカー位置のコレクションを指定すると定義されます。

4Dフォームエディターでは、Altキーを押しながら適切なコントロールマーカーをクリックして、追加のコントロールラインを作成します。

フォームは常に、ヘッダー、詳細、ブレーク (レベル0)、およびフッターエリアを持った状態で作成されます。

ブレークレベル0 は、レポート対象の全レコードが印刷された後に実行されます。追加のブレークマーカーはそれぞれ "ブレークレベル1"、"ブレークレベル2" などと呼ばれます。

レベル1のブレークは一番目のソートフィールドでグループ化されたレコードが印刷された後に発生します。

ラベル	説明	実行タイミング
B1	ブレークレベル1	一番目のソートフィールド印刷後
B2	ブレークレベル2	二番目のソートフィールド印刷後
B3	ブレークレベル3	三番目のソートフィールド印刷後

追加のヘッダーはブレークに関連付けられます。レベル1ヘッダーは、最初のソートフィールドによりグループ化されたレコードが印刷される直前に印刷されます。

ラベル	説明	実行タイミング
H1	ヘッダーレベル1	一番目のソートフィールド印刷後
H2	ヘッダーレベル2	二番目のソートフィールド印刷後
H3	ヘッダーレベル3	三番目のソートフィールド印刷後

ブレーク処理を開始するために `Subtotal` コマンドを使用する場合、ソート順に従って作成されるすべてのブレークレベル数から 1 マイナスした数のブレークエリアを作成すべきです。ブレークエリアに何も印刷する必要がない場合、マーカーを移動してその高さを 0 にします。ブレークエリアの数よりも多いフィールド数でソートすると、印刷時に最後のブレークエリアが繰り返されます。

メニュー

連結メニューバー

メニューバーをフォームに割り当てると、アプリケーションモードでフォームが表示されたときにカレントメニューバーの右側に追加されます。

メニュー命令が選択されると、`On Menu Selected` イベントがフォームメソッドに送られます。コード内では、`Menu selected` コマンドを使って、選択されたメニューを確認することができます。

フォームのメニューバーとカレントメニューバーが同じ場合、追加はされません。

フォームメニューバーは、入力および出力フォームの両方で動作します。

JSON 文法

名称	データタイプ	とりうる値
menuBar	string	メニューバーの名称

印刷

Settings

フォーム毎に用紙設定をおこなうことができます。この機能は、フォームエディターで印刷ページの境界を表示するのに便利です。

互換性に関する注意: アプリケーションモードでフォームを印刷するときにこれらの設定が考慮されたとしても、プラットフォームおよびドライバー依存性に関する制約から、フォームの印刷設定を保存するのにこの機能を使用することは推奨されません。より強力な `Print settings to BLOB / BLOB to print settings` を使用することが強く推奨されています。

次の印刷設定が変更できます:

- 用紙サイズ
- 用紙の向き
- 拡大縮小

利用可能なオプションはシステムの設定により異なります。

JSON 文法

名称	データタイプ	とりうる値
pageFormat	object	利用可能なプロパティ: <code>paperName</code> , <code>paperWidth</code> , <code>paperHeight</code> , <code>orientation</code> , <code>scale</code>
paperName	string	"A4", "レター"...
paperWidth	string	<code>paperName</code> という名前の用紙が見つからなかった場合に使用されます。単位の明記が必要です: <code>pt</code> , <code>in</code> , <code>mm</code> , <code>cm</code> .
paperHeight	string	<code>paperName</code> という名前の用紙が見つからなかった場合に使用されます。単位の明記が必要です: <code>pt</code> , <code>in</code> , <code>mm</code> , <code>cm</code> .
orientation	string	"landscape" (デフォルトは "portrait")
scale	number	最小値: 0

ウィンドウサイズ

固定高さ

このオプションを選択するとウィンドウの高さが固定され、ユーザーは変更できなくなります。

このオプションを選択しない場合、フォームのウィンドウの高さを変更することができます。この場合 [最小高さと最大高さ](#) 入力エリアを使用し、変更できるサイズを制限できます。

JSON 文法

名称	データタイプ	とりうる値
windowSizingY	string	"fixed", "variable"

固定幅

このオプションを選択するとウィンドウの幅が固定され、ユーザーは変更できなくなります。

このオプションを選択しない場合、フォームのウィンドウの高さを変更することができます。この場合 [最小幅と最大幅](#) 入力エリアを使用し、変更できるサイズを制限できます。

JSON 文法

名称	データタイプ	とりうる値
windowSizingX	string	"fixed", "variable"

最大高さ, 最小高さ

[固定高さ](#) オプションが設定されていない場合の、サイズ変更可能なフォームウィンドウの最大および最小の高さ (ピクセル単位) です。

JSON 文法

名称	データタイプ	とりうる値
windowMinHeight	number	整数値
windowMaxHeight	number	整数値

最大幅, 最小幅

[固定幅](#) オプションが設定されていない場合の、サイズ変更可能なフォームウィンドウの最大および最小の幅 (ピクセル単位) です。

JSON 文法

名称	データタイプ	とりうる値
windowMinWidth	number	整数値
windowMaxWidth	number	整数値

4D フォームオブジェクトについて

フォーム上のオブジェクトを編集することで、アプリケーションのフォームを構築しカスタマイズします。オブジェクトの追加・配置、オブジェクトプロパティの設定、入力規制の追加によるビジネスルールの強化、操作に対して自動実行されるオブジェクトメソッドの追加などをおこなうことができます。

アクティブオブジェクトとスタティックオブジェクト

4D フォームでは多くのビルトイン アクティブ オブジェクトおよび スタティック オブジェクトが提供されています:

- アクティブオブジェクト はインターフェース機能やデータベーススクを実行します。アクティブオブジェクトの種類は、入力フィールド、コンボボックス、ドロップダウンリスト、ピクチャーボタンなど様々ですが、いずれもデータを表示したり、メモリに一時保存したり、ダイアログボックスを開く・レポートを印刷する・バックグラウンドプロセスを開始するなどの動作を実行したりします。
- スタティックオブジェクト (線、枠、背景ピクチャー等) は一般的に、フォームのアピアランスやラベル、グラフィックインターフェースを設定するために使用されます。アクティブオブジェクトと異なり、スタティックオブジェクトには変数や式が割り当てられません。しかし、スタティックオブジェクトにダイナミックオブジェクトを挿入することは可能です。

オブジェクトの操作

4D フォームオブジェクトの追加や編集は次の方法でおこなえます:

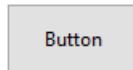
- **フォームエディター**: ツールバーからオブジェクトをフォーム上にドラッグ & ドロップします。フォームに配置したら、そのオブジェクトのプロパティをプロパティリストから編集できます。
詳細については [フォームの作成](#) を参照ください。
- 4D ランゲージ: **オブジェクト(フォーム)** テーマのコマンド (`OBJECT DUPLICATE` や `OBJECT SET FONT STYLE` など) を使って、フィームオブジェクトを作成・定義することができます。
- ダイナミックフォーム内の JSON コード: JSON を使ってプロパティを定義します。`type` プロパティでオブジェクトタイプを定義し、提供されている他のプロパティのうち必要なものを設定します。詳細については [ダイナミックフォーム](#) ページを参照ください。

次はボタンオブジェクトの例です:

```
{ "type": "button", "style": "bevel", "text": "OK", "action": "Cancel", "left": 60, "top": 160, "width":
```

ボタン

ボタンとは、ユーザーのクリック操作に応じて実行されるアクション（例 データベーススクやインターフェース機能）を割り当てることのできるアクティブオブジェクトです。



割り当てられたスタイルやアクションに応じて、ボタンはさまざまな役割を果たします。たとえば、ユーザーがアンケートやフォーム内を移動したり、選択したりといった操作を可能にします。設定によって、ボタンをワンクリックすることでコマンド実行することもできれば、複数回クリックすることで望む結果を得られるようにすることもできます。

ボタンの使用

ボタンには、あらかじめ設定されている [標準アクション](#) またはオブジェクトメソッドを割り当てることができます。典型的なアクションの例は、レコードの受け入れ・削除・編集キャンセルのほか、データのコピー / ペースト、複数ページあるフォームにおけるページ移動、サブフォームのレコード操作、テキストエリアのフォント属性の操作、などです。

フォーム実行時、標準アクションが設定されたボタンは必要に応じてグレー表示されます。たとえば、あるテーブルの1番目のレコードが表示されていると、先頭レコード（`firstRecord`）標準アクションがついたボタンはグレー表示されます。

標準アクションとして提供されていない動作をボタンに実行させたい場合には、標準アクションのフィールドは空欄にしておき、ボタンのアクションを指定するオブジェクトメソッドを書きます オブジェクトメソッドの作成や割り当て方法についての詳細は [オブジェクトメソッドを使用する](#) を参照ください。通常は、イベントテーマで `On Clicked` イベントを有効にして、ボタンのクリック時にのみメソッドを実行します。どのタイプのボタンにもメソッドを割り当てることができます。

ボタンに関連付けられた変数（[variable](#) 属性）は、デザインモードやアプリケーションモードでフォームが初めて開かれるときに自動で 0 に初期化されます。ボタンをクリックすると、変数の値は 1 になります。

ボタンには標準アクションとメソッドの両方を割り当てることもできます。この場合、標準アクションによってボタンが無効化されていなければ、標準アクションより先にメソッドが実行されます。

ボタンスタイル

ボタンスタイルは、ボタンの外観を制御すると同時に、提供されるプロパティも決定します。ボタンには、あらかじめ定義されたスタイルやポップアップメニューを割り当てることができます。これらのプロパティや動作を組み合わせることで、多数のバリエーションが得られます。

スタイルによって [提供されるプロパティ](#) は異なりますが、大多数のボタンは構造上 同じです。違いは、関連づけられた変数の処理にあります。

次の既定スタイルが提供されています：

通常

通常スタイルのボタンは、標準的なシステムボタンで（長方形にラベルが付いたもの）、ユーザークリックに応じてコードを実行します。



通常ボタンは、明るいグレーの背景に中央配置のラベルがデフォルトで付いています。Windowsの場合は、通常ボタンの上にマウスオーバーすると、境界線色と背景色が変わります。また、プラットフォームによって表現が異なりますが、クリック時にも背景色が変わるなどして、物理的なボタンを模倣します。

JSON 例：

```

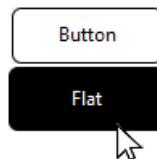
"myButton": {
    "type": "button", // オブジェクトタイプ
    "style": "regular", // ボタンスタイル
    "defaultButton": "true" // デフォルトボタン
    "text": "OK", // タイトル
    "action": "Cancel", // アクション
    "left": 60, // フォーム上の座標 (左)
    "top": 160, // フォーム上の座標 (上)
    "width": 100, // 幅
    "height": 20 // 高さ
}

```

[デフォルトボタン](#) プロパティが提供されているのは、通常スタイルとフラットスタイルのみです。

フラット

フラットスタイルのボタンは、標準的なシステムボタンで（長方形にラベルが付いたもの）、ユーザークリックに応じてコードを実行します。



フラットボタンは、白の背景に中央配置のラベルがデフォルトで付いており、角が丸くなっています。フラットボタンのグラフィック的な装飾は最小限であるため、印刷されるフォームでの使用に適しています。

JSON 例:

```

"myButton": {
    "type": "button",
    "style": "flat",
    "defaultButton": "true"
    "text": "OK",
    "action": "Cancel",
    "left": 60,
    "top": 160,
    "width": 100,
    "height": 20
}

```

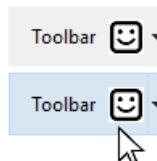
[デフォルトボタン](#) プロパティが提供されているのは、通常スタイルとフラットスタイルのみです。

ツールバー

ツールバースタイルのボタンは、主としてツールバーで使用するためのものです。このスタイルには、複数の選択肢を提示するためのポップアップメニュー（逆三角形で示されます）を追加するオプションがあります。

ツールバー ボタンは、透明の背景に中央配置のラベルがデフォルトで付いています。ボタンにマウスオーバーしたときの表示は OS によって異なります：

- Windows - ボタンがハイライト表示されます。"ポップアップメニューあり" プロパティを使用していると、ボタンの右側中央に三角形が表示されます。



- macOS - ボタンはハイライト表示されません。"ポップアップメニューあり" プロパティを使用していると、ボタンの右側に三角形が表示されます。

JSON 例:

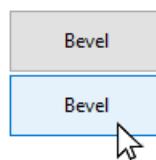
```
"myButton": {  
    "type": "button",  
    "style": "toolbar",  
    "text": "OK",  
    "popupPlacement": "separated",  
    "action": "Cancel",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

ベベル

ベベルスタイルは [通常](#) スタイルの外観 (四角にラベル) に [ツールバー](#) スタイルのポップアップメニューを追加可能にしたものです。

ベベルボタンは、明るいグレーの背景に中央配置のラベルがデフォルトで付いています。ボタンにマウスオーバーしたときの表示は OS によって異なります:

- Windows - ボタンがハイライト表示されます。"ポップアップメニューあり" プロパティを使用していると、ボタンの右側に三角形が表示されます。



- macOS - ボタンはハイライト表示されません。"ポップアップメニューあり" プロパティを使用していると、ボタンの右側に三角形が表示されます。

JSON 例:

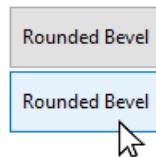
```
"myButton": {  
    "type": "button",  
    "style": "bevel",  
    "text": "OK",  
    "popupPlacement": "linked",  
    "action": "Cancel",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

角の丸いベベル

角の丸いベベルスタイルは [ベベル](#) スタイルとほぼ同一ですが、OSによっては角が丸く表示されます。ベベルスタイルと同様に、角の丸いベベルスタイルは [通常](#) スタイルの外観 (四角にラベル) に [ツールバー](#) スタイルのポップアップメニューを追加可能にしたものです。

角の丸いベベルボタンは、明るいグレーの背景に中央配置のラベルがデフォルトで付いています。ボタンにマウスオーバーしたときの表示は OS によって異なります:

- Windows - ベベルボタンと同じです。"ポップアップメニューあり" プロパティを使用していると、ボタンの右側に三角形が表示されます。



- macOS - 角が丸くなっています。"ポップアップメニューあり" プロパティを使用していると、ボタンの右側に三角形が表示されます。

JSON 例:

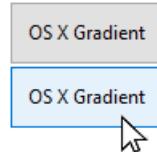
```
"myButton": {  
    "type": "button",  
    "style": "roundedBevel",  
    "text": "OK",  
    "popupPlacement": "none" /  
    "action": "Cancel",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

OS Xグラデーション

OS Xグラデーションスタイルは [ベベル](#) スタイルとほぼ同一です。ベベルスタイルと同様に、OS Xグラデーションスタイルは [通常](#) スタイルの外観 (四角にラベル) に [ツールバー](#) スタイルのポップアップメニューを追加可能にしたものです。

OS Xグラデーションボタンは、明るいグレーの背景に中央配置のラベルがデフォルトで付いています。ボタンにマウスオーバーしたときの表示は OS によって異なります:

- *Windows* - ベベルボタンと同じです。"ポップアップメニューあり" プロパティを使用していると、ボタンの右側に三角形が表示されます。



- *macOS* - 2トーンのシステムボタンです。"ポップアップメニューあり" プロパティを使用していると、ボタンの右側に三角形が表示されます。

JSON 例:

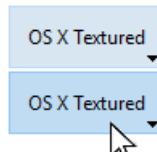
```
"myButton": {  
    "type": "button",  
    "style": "gradientBevel",  
    "text": "OK",  
    "popupPlacement": "linked"  
    "action": "Cancel",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

OS Xテクスチャ

OS Xテクスチャスタイルは [ベベル](#) スタイルとほぼ同一ですが、サイズがより小さいです (最大サイズは macOS の標準的なシステムボタンのサイズです)。ベベルスタイルと同様に、OS Xテクスチャスタイルは [通常](#) スタイルの外観 (四角にラベル) に [ツールバー](#) スタイルのポップアップメニューを追加可能にしたものです。

デフォルトで、OS Xテクスチャボタンの外観は次の通りです:

- *Windows* - 明るいブルーの背景に中央配置のラベルが付いた標準のシステムボタンです。Vistaにおいては透明になる特別機能を持っています。



- macOS - 灰色のグラデーションを表示する標準のシステムボタンです。高さは定義済みで、変更できません。

JSON 例:

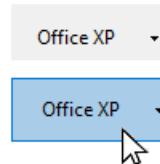
```
"myButton": {  
    "type": "button",  
    "style": "texturedBevel",  
    "text": "OK",  
    "popupPlacement": "separated",  
    "action": "Cancel",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

Office XP

ペベルスタイルと同様に、Office XPスタイルは [通常](#) スタイルの外観 (四角にラベル) に [ツールバー](#) スタイルの透過性を加え、ポップアップメニューを追加可能にしたものです。

Office XPボタンの反転表示と背景のカラーはシステムカラーに基づいています。ボタンにマウスオーバーしたときの表示は OS によって異なります:

- Windows - マウスオーバー時にのみ背景が表示されます。



- macOS - 背景は常に表示されます。

JSON 例:

```
"myButton": {  
    "type": "button",  
    "style": "office",  
    "text": "OK",  
    "popupPlacement": "none",  
    "action": "Cancel",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

ヘルプ

このスタイルはシステム標準のヘルプボタンを表示するために使用します。デフォルトで、ヘルプボタンは丸の中に表示されたハテナマーク (疑問符) です。



JSON 例:

```
"myButton": {  
    "type": "button",  
    "style": "help",  
    "text": "OK",  
    "dropping": "custom",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

ヘルプボタンは次の基本プロパティを持ちません: 状態の数、ピクチャーパス名、タイトル/ピクチャー位置。

サークル

サークルスタイルのボタンは、円形のシステムボタンとして表示されます。このボタンスタイルは macOS 用に用意されています。



Windows の場合、サークルは表示されません。

JSON 例:

```
"myButton": {  
    "type": "button",  
    "style": "circular",  
    "text": "OK",  
    "dropping": "custom",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

カスタム

カスタムスタイルのボタンは、背景ピクチャーを使用できるほか、さまざまな追加パラメーターを管理することができます (アイコンオフセットやマージン)。



JSON 例:

```
"myButton": {  
    "type": "button",  
    "style": "custom",  
    "text": "",  
    "customBackgroundPicture": "/RESOURCES/bkgnd.png",  
    "icon": "/RESOURCES/custom.png",  
    "textPlacement": "center",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

プロパティ一覧

すべてのボタンは次の基本プロパティを共有します:

タイプ - オブジェクト名 - 変数あるいは式 - タイトル - CSSクラス - ボタンスタイル - ピクチャーパス名(1) - 状態の数(1) - タイトル/ピクチャー位置(1) - 左 - 上 - 右 - 下 - 幅 - 高さ - 横方向サイズ変更 - 縦方向サイズ変更 - フォーカス可 - ショートカット - 表示状態 - レンダリングしない - 境界線スタイル - フォント - フォントサイズ - 太字 - イタリック - 下線 - フォントカラー - ヘルプTips - 標準アクション - ドロップ有効

(1) ヘルプ スタイルは除外

ボタンスタイル に応じて、次の追加プロパティが使用できます:

- 背景パス名 - アイコンオフセット - 横方向マージン - 縦方向マージン (カスタムスタイル)
- デフォルトボタン (通常、フラット)
- ポップアップメニューあり (ツールバー、ベベル、角の丸いベベル、OS X グラデーション、OS X テクスチャー、Office XP、サークル、カスタム)

ボタングリッド

ボタングリッドは透明なオブジェクトであり、画像の最前面に配置されます。おもに、列と行の配列を表現する画像との組み合わせで使用します。ユーザーがグラフィック上でクリックすると、そこが凹んだように描画されます。

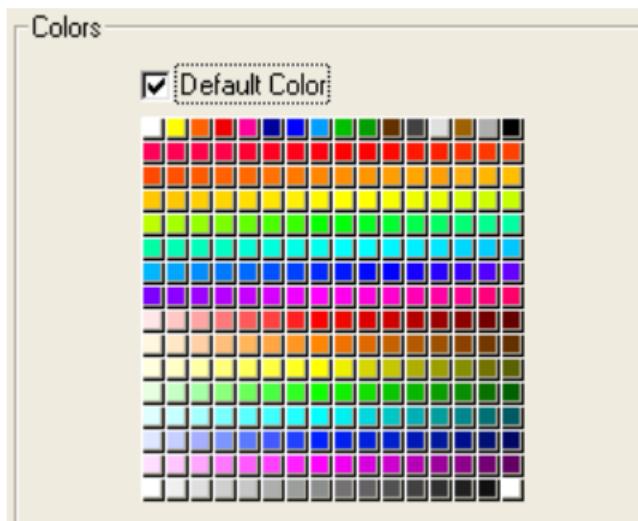


ボタングリッドのオブジェクトを使用すると、グラフィック上でユーザーがクリックした場所を判別することができます。オブジェクトメソッドでは `On Clicked` イベントを使用し、クリックされた場所に応じて適切な動作を実行します。

ボタングリッドの作成

ボタングリッドを作成するには、背景グラフィックをフォームに追加し、その最前面にボタングリッドを配置します。"行列数" テーマの [行](#) と [列](#) に行数と列数を指定します。

4D では、カラーパレットにボタングリッドが使用されています：



ボタングリッドの使用

グリッド上のボタンには、左上から右下に向けて番号が振られます。上の例で、グリッドは 16列×16行で構成されています。左上にあるボタンはクリックされると 1 を返します。2行目の右端にある赤いボタンが選択されると、ボタングリッドは 32 を返します。選択されたボタンがなければ、0が返されます。

ページ指定アクション

ボタングリッドにページ指定用の `gotoPage` 標準アクションを割り当てることができます。この標準アクションを設定すると、4D はボタングリッドで選択されたボタンの番号に相当するフォームページを自動的に表示します。たとえばグリッド上の 10 番目のボタンを選択すると、4D は現在のフォームの 10 ページ目を表示します（存在する場合）。

プロパティ一覧

タイプ - オブジェクト名 - 変数あるいは式 - CSSクラス - 行 - 列 - 左 - 上 - 右 - 下 - 幅 - 高さ - 横方向サイズ変更 - 縦方向サイズ変更 - 表示状態 - 境界線スタイル - ヘルプTips - 標準アクション

チェックボックス

チェックボックスはボタンの一種で、バイナリ (true-false) データの入力や表示をおこないます。基本的に、チェックボックスの状態は選択または未選択のいずれかになりますが、[3つめの状態](#) を定義することもできます。



チェックボックスは、メソッドまたは [標準アクション](#) を使って管理します。チェックボックスが選択されると、チェックボックスに割り当てられたメソッドが実行されます。他のボタンと同じように、フォームが初めて開かれると、チェックボックスの変数は 0 に初期化されます。

チェックボックスは小さな四角形の右側にテキストを表示します。このテキストはチェックボックスの [タイトル](#) プロパティで設定します。タイトルには、XLIFF 参照を入れることもできます ([付録 B: XLIFFアーキテクチャー 参照](#))。

チェックボックスの使用

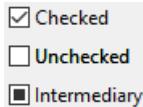
チェックボックスには整数型またはブール型の [変数あるいは式](#) を設定することができます。

- 整数型: チェックボックスが選択されると、変数の値は 1 になります。チェックボックスが選択されていない場合の値は 0 です。チェックボックスが 3 番目の状態 (後述参照) のとき、変数値は 2 になります。
- ブール型: チェックボックスが選択されると、変数の値は `true` になります。チェックボックスが選択されていない場合の値は `False` です。

フォーム上のすべてのチェックボックスは選択/未選択のいずれかの状態にすることができます。チェックボックスを複数使用することで、複数の候補を同時に選択できるようになります。

スリーステートチェックボックス

[ボタンスタイル](#) が [通常](#) および [フラット](#) タイプのチェックボックスは 3 番目の状態を受け入れます。この 3 番目の状態は中間的な状態であり、一般的には表示のために用いられます。たとえば、選択されたオブジェクトが複数あるうち、一部のオブジェクトにのみ特定のプロパティが存在することを表すのに使用されます。



この 3 番目の状態を有効にするには [スリーステート](#) プロパティを選択します。

このプロパティは、数値型の [変数あるいは式](#) に関連付けられた通常およびフラットスタイルのチェックボックスに対してのみ使用できます。ブール型のチェックボックスは [スリーステート](#) プロパティを利用できません (ブール式には中間状態が存在しません)。

チェックボックスが 3 番目の状態になると、チェックボックスに関連付けられた変数は値 2 を返します。

スリーステートチェックボックスは入力モードにおいて、チェックなし / チェック / 中間状態 / チェックなし、という順に状態表示を切り替えます。一般的にこの中間状態は入力モードではあまり役に立たないため、2 という値になった場合は、コード上で変数の値を強制的に 0 に設定し、チェックされた状態からチェックなしの状態へ直接移行します。

標準アクションの使用

チェックボックスに [標準アクション](#) を割り当てることで、テキストエリアの属性を管理することができます。たとえば、`fontBold` 標準アクションを選択すると、ランタイムにおいてそのチェックボックスは、カレントエリア内で選択されたテキストの "bold" 属性を管理します。

`true/false` ステータスで表すことのできるアクション ("checkable" アクション) のみがサポートされます:

サポートされるアクション	使用条件 (あれば)
avoidPageBreakInsideEnabled	4D Write Proエリアのみ
fontItalic	
fontBold	
fontLinethrough	
fontSubscript	4D Write Proエリアのみ
fontSuperscript	4D Write Proエリアのみ
fontUnderline	
font/showDialog	macOS のみ
htmlWYSIWIGEnabled	4D Write Proエリアのみ
section/differentFirstPage	4D Write Proエリアのみ
section/differentLeftRightPages	4D Write Proエリアのみ
spell/autoCorrectionEnabled	
spell/autoDashSubstitutionsEnabled	macOS のみ
spell/autoLanguageEnabled	macOS のみ
spell/autoQuoteSubstitutionsEnabled	macOS のみ
spell/autoSubstitutionsEnabled	
spell/enabled	
spell/grammarEnabled	macOS のみ
spell/showDialog	macOS のみ
spell/visibleSubstitutions	
visibleBackground	4D Write Proエリアのみ
visibleFooters	4D Write Proエリアのみ
visibleHeaders	4D Write Proエリアのみ
visibleHiddenChars	4D Write Proエリアのみ
visibleHorizontalRuler	4D Write Proエリアのみ
visiblePageFrames	4D Write Proエリアのみ
visibleReferences	
widowAndOrphanControlEnabled	4D Write Proエリアのみ

これらのアクションについての詳細は、[標準アクション](#) の章を参照してください。

チェックボックスのボタンスタイル

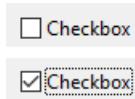
チェックボックスの [ボタンスタイル](#) は、チェックボックスの外観を制御すると同時に、提供されるプロパティをも決定します。チェックボックスには、あらかじめ定義されたスタイルを割り当てることができます。これらのプロパティや動作を組み合わせることで、多数のバリエーションが得られます。

スタイルによって [提供されるプロパティ](#) は異なりますが、大多数のチェックボックスは 構造上 同じです。違いは、関連づけられた変数の処理にあります。

次の既定スタイルが提供されています:

通常

通常スタイルのチェックボックスは、標準的なシステムチェックボックス（四角形にタイトルが付いたもの）です。

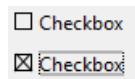


JSON 例:

```
"myCheckBox": {  
    "type": "checkbox",  
    "style": "regular",  
    "text": "Cancel",  
    "action": "Cancel",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
    "dataSourceTypeHint": "boolean"  
}
```

フラット

フラットスタイルのチェックボックスでは、装飾が最小限に抑えられています。このグラフィック的特性により、フラットスタイルは印刷フォームでの使用に適しています。



JSON 例:

```
"myCheckBox": {  
    "type": "checkbox",  
    "style": "flat",  
    "text": "Cancel",  
    "action": "cancel",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

ツールバー ボタン

チェックボックスのツールバー ボタンスタイルは、主としてツールバーで使用するためのものです。

ツールバー ボタンスタイルは、透明の背景にラベルが付いています。また、通常は [4つの状態を持つ画像](#) が関連付けられます。

チェックなし / チェック / ハイライト状態の例です:



JSON 例:

```
"myCheckBox": {  
    "type": "checkbox",  
    "style": "toolbar",  
    "text": "Checkbox",  
    "icon": "/RESOURCES/File.png",  
    "iconFrames": 4  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

ベベル

ベベルスタイルはデフォルトでは、通常ボタンのような外観に、チェックボックスの [ツールバー ボタン](#) スタイルの機能を組み合わせたものです。

ベベルスタイルは、明るいグレーの背景にラベルが付いています。また、通常は [4つの状態を持つ画像](#) が関連付けられます。

チェックなし / チェック / ハイライト状態の例です：



JSON 例：

```
"myCheckBox": {  
    "type": "checkbox",  
    "style": "bevel",  
    "text": "Checkbox",  
    "icon": "/RESOURCES/File.png",  
    "iconFrames": 4  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

角の丸いベベル

角の丸いベベルスタイルは [ベベル](#) スタイルとほぼ同一ですが、OSによっては角が丸く表示されます。ベベルスタイルと同様に、角の丸いベベルスタイルは通常ボタンのような外観に、チェックボックスの [ツールバー ボタン](#) スタイルの機能を組み合わせたものです。

角の丸いベベルスタイルは、明るいグレーの背景にラベルが付いています。また、通常は [4つの状態を持つ画像](#) が関連付けられます。

macOS の例：



Windows 上では、角の丸いベベルスタイルは [ベベル](#) スタイルと同じです。

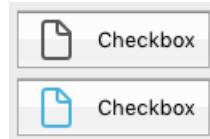
JSON 例:

```
"myCheckBox": {  
    "type": "checkbox",  
    "style": "roundedBevel",  
    "text": "Checkbox",  
    "icon": "/RESOURCES/File.png",  
    "iconFrames": 4  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

OS Xグラデーション

OS Xグラデーションスタイルは [ベベル](#) スタイルとほぼ同一です。ベベルスタイルと同様に、OS Xグラデーションスタイルは通常ボタンのような外観に、チェックボックスの [ツールバー](#)ボタン スタイルの機能を組み合わせたものです。

OS X グラデーションスタイルは明るいグレーの背景にラベルが付いています。macOS 上では2トーンのシステムボタンとして表示されることがあります。また、通常は [4つの状態を持つ画像](#) が関連付けられます。



Windows 上では、このスタイルは [ベベル](#) スタイルと同じです。

JSON 例:

```
"myCheckBox": {  
    "type": "checkbox",  
    "style": "gradientBevel",  
    "text": "Checkbox",  
    "icon": "/RESOURCES/File.png",  
    "iconFrames": 4  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

OS Xテクスチャー

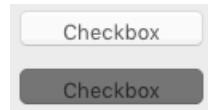
OS Xテクスチャースタイルは [ベベル](#) スタイルと似ていますが、サイズがより小さいです（最大サイズは macOS の標準的なシステムボタンのサイズです）。ベベルスタイルと同様に、OS Xテクスチャースタイルは通常ボタンのような外観に、チェックボックスの [ツールバー](#)ボタン スタイルの機能を組み合わせたものです。

デフォルトで、OS Xテクスチャースタイルの外観は次の通りです:

- Windows - 明るいブルーの背景に中央配置のラベルが付いた標準のシステムボタンです。



- macOS - 標準のシステムボタンです。高さは定義済みで、変更できません。



JSON 例:

```
"myCheckBox": {  
    "type": "checkbox",  
    "style": "texturedBevel",  
    "text": "Checkbox",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

Office XP

Office XPスタイルは通常ボタンのような外観に、ツールバー ボタン スタイルの動作を組み合わせたものです。

Office XPスタイルのチェックボックスの反転表示と背景のカラーはシステムカラーに基づいています。チェックボックスにマウスオーバーしたときの表示は OS によって異なります:

- Windows - マウスオーバー時にのみ背景が表示されます。チェックなし / チェック / ハイライト状態の例です:



- macOS - 背景は常に表示されます。チェックなし / チェック状態の例です:



JSON 例:

```
"myCheckBox": {  
    "type": "checkbox",  
    "style": "office",  
    "text": "Checkbox",  
    "action": "fontBold",  
    "icon": "/RESOURCES/File.png",  
    "iconFrames": 4  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

折りたたみ/展開

このチェックボックススタイルは標準の折りたたみ/展開アイコンを表示するのに使用します。これらは階層リストで使用されます。

- Windows - [+] または [-] のように表示されます。



- macOS - 右や下を指す三角として表示されます。



JSON 例:

```
"myCheckBox": {  
    "type": "checkbox",  
    "style": "disclosure",  
    "method": "mCollapse",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

開示ボタン

開示ボタンスタイルが適用されたチェックボックスは macOS および Windowsにおいて、詳細情報の表示/非表示にするのに使われる標準的な開示ボタンとして描画されます。値が 0 のときにはボタンの矢印が下向き、値が 1 のときは上向きになります。

- Windows



- macOS



JSON 例:

```
"myCheckBox": {  
    "type": "checkbox",  
    "style": "roundedDisclosure",  
    "method": "m_disclose",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

カスタム

カスタムスタイルのチェックボックスは、背景ピクチャーを使用できるほか、さまざまな追加パラメーターを管理することができます:

- [背景パス名](#)
- [アイコンオフセット](#)
- [横方向マージン](#) と [縦方向マージン](#)

カスタムチェックボックスには通常、[4つの状態を持つ画像](#) が関連付けられ、これは同じく [4つの状態を持つ 背景ピクチャー](#) と同時に使用することができます。

JSON 例:

```
"myCheckbox": {  
    "type": "checkbox",  
    "style": "custom",  
    "text": "OK",  
    "icon": "/RESOURCES/smiley.jpg",  
    "iconFrame": 4,  
    "customBackgroundPicture": "/RESOURCES/paper.jpg",  
    "iconOffset": 5, // クリック時のアイコンオフセット  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20,  
    "customBorderX": 20,  
    "customBorderY": 5  
}
```

プロパティ一覧

すべてのチェックボックスは次の基本プロパティを共有します:

[タイプ](#) - [オブジェクト名](#) - [変数あるいは式](#) - [式の型](#) - [タイトル](#) - [値を記憶](#) - [CSSクラス](#) - [ボタンスタイル](#) - [左](#) - [上](#) - [右](#) - [下](#) - [幅](#) - [高さ](#) - [横方向サイズ変更](#) - [縦方向サイズ変更](#) - [入力可](#) - [フォーカス可](#) - [ショートカット](#) - [表示状態](#) - [フォント](#) - [フォントサイズ](#) - [太字](#) - [イタリック](#) - [下線](#) - [フォントカラー](#) - [ヘルプTips](#) - [標準アクション](#)

[ボタンスタイル](#) に応じて、次の追加プロパティが使用できます:

- [背景パス名](#) - [アイコンオフセット](#) - [横方向マージン](#) - [縦方向マージン](#) (カスタムスタイル)
- [スリーステート](#) (通常、フラット)
- [ピクチャーパス名](#) - [状態の数](#) - [タイトル/ピクチャー位置](#) (ツールバーボタン、ベベル、角の丸いベベル、OS X グラデーション、OS X テクスチャー、Office XP、カスタム)

コンボボックス

コンボボックスは [ドロップダウンリスト](#) と似ていますが、キーボードから入力されたテキストを受けいれる点と、二つの追加オプションがついている点が異なります。



基本的にコンボボックスは入力エリアと同じように取り扱い、オブジェクト、配列、または選択リストを一連のデフォルト値として使用します。

コンボボックスの操作

入力エリアへの入力内容は、その他の入力フォームオブジェクトと同様に `On Data Change` イベントを使用して管理します。

コンボボックスの初期化方法は、[ドロップダウンリスト](#) とまったく同じで、オブジェクト、配列、または選択リストを使用できます。

オブジェクトの使用

この機能は 4Dプロジェクトでのみ利用可能です。

コンボボックスのデータソースとして、[コレクション](#) を内包した [オブジェクト](#) を使用できます。このオブジェクトには、次のプロパティが格納されていく必要があります:

プロパティ	タイプ	説明
<code>values</code>	コレクション	必須 - スカラー値のコレクション。すべての同じ型の値でなくてはなりません。サポートされている型: <ul style="list-style-type: none">● 文字列● 数値● 日付● 時間 空、または未定義の場合、コンボボックスは空になります
<code>currentValue</code>	Collection要素と同じ	ユーザーによる入力値

オブジェクトにその他のプロパティが含まれている場合、それらは無視されます。

ユーザーによってコンボボックスに入力されたテキストは、オブジェクトの `currentValue` プロパティが受け取ります。

配列の使用

配列を初期化する方法については、[ドロップダウンリスト](#) ページの [配列の使用](#) を参照ください。

ユーザーによってコンボボックスに入力されたテキストは、配列の 0番目の要素が受け取ります。

選択リストの使用

入力エリア (列挙型のフィールドまたは変数) の管理のためにコンボボックスを使用したい場合、フィールドまたは変数をフォームオブジェクトのデータソースとして直接参照することができます。これにより列挙型のフィールド/変数を容易に管理できるようになります。

階層リストの場合、第一階層の値のみが表示・選択できます。

コンボボックスにフィールドや変数を関連付けるには、フォームオブジェクトの [変数あるいは式](#) 欄にフィールドまたは変数の名前を直接入力します。

フォームを実行すると、4D が自動的に入力中または表示中のコンボボックスの状態を管理します。ユーザーが値を選択すると、その値はフィールドに保存され、このフィールドの値はフォームが表示されたときにコンボボックスに表示されます：

詳細については、[ドロップダウンリスト](#) ページの [選択リストの使用](#) を参照ください。

オプション

コンボボックスタイプのオブジェクトには、2つの専用オプションがあります：

- [自動挿入](#)：このオプションがチェックされていると、オブジェクトに関連付けられたリストにない値をユーザーが入力した場合に、その値が自動的にデータソースに追加されます。
- [除外リスト](#) (除外される値のリスト)：除外される値のリストを関連付けることができます。ユーザーがこのリストに含まれる値を入力したとき、その入力は自動的に却下され、エラーメッセージが表示されます。

[指定リスト](#) はコンボボックスには割り当てることはできません。ユーザーインターフェースにおいて、オブジェクト内にいくつかの指定された値を表示したいときには、[ドロップダウンリスト](#) のオブジェクトを使用して下さい。

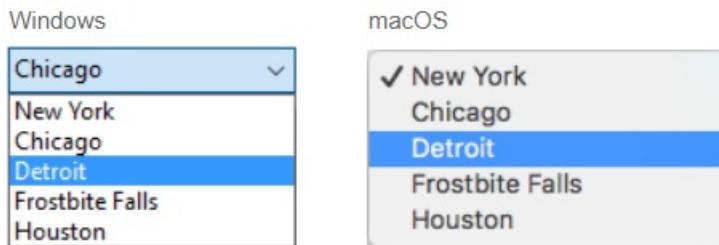
プロパティ一覧

[タイプ](#) - [オブジェクト名](#) - [変数あるいは式](#) - [式の型](#) - [CSSクラス](#) - [選択リスト](#) - [左](#) - [上](#) - [右](#) - [下](#) - [幅](#) - [高さ](#) - [横方向サイズ変更](#) - [縦方向サイズ変更](#) - [文字フォーマット](#) - [日付フォーマット](#) - [時間フォーマット](#) - [表示状態](#) - [フォント](#) - [フォントサイズ](#) - [太字](#) - [イタリック](#) - [下線](#) - [フォントカラー](#) - [ヘルプTips](#)

ドロップダウンリスト

ドロップダウンリストは、ユーザーがリストから選択をおこなえるようにするためのフォームオブジェクトです。ドロップダウンリストに表示される項目は、オブジェクト、配列、選択リスト、または標準アクションを用いて管理します。

macOSにおいては、ドロップダウンリストは "ポップアップメニュー" とも呼ばれます。どちらの名前も同じタイプのオブジェクトを指します。次の例に示すように、ドロップダウンリストの外観はプラットフォームによって若干異なります：



ドロップダウンリストの種類

それぞれに特有の機能を持つ、複数タイプのドロップダウンリストを作成することができます。タイプを定義するには、プロパティリストで適切な式の型とデータタイプの値を選択するか、それらに相当する JSON を指定します。

タイプ	機能	式の型/ 式タイプ	データタイプ	JSON 定義
オブジェクト	コレクションに基づく	オブジェクト	Numeric, Text, Date, または Time	<code>dataSourceTypeHint: object + numberFormat: <format> または textFormat: <format> または dateFormat: <format> または timeFormat: <format></code>
配列	配列に基づく	配列	Numeric, Text, Date, または Time	<code>dataSourceTypeHint: arrayNumber または arrayText または arrayDate または arrayTime</code>
選択リスト(値を保存)	選択リストに基づく(標準)	リスト	選択された項目値	<code>dataSourceTypeHint: text + saveAs: value</code>
選択リスト(参照を保存)	選択リストに基づく(項目の位置を保存)	リスト	選択された項目参照	<code>dataSourceTypeHint: integer + saveAs: reference</code>
階層型選択リスト	階層型の表示が可能	リスト	リスト参照	<code>dataSourceTypeHint: integer</code>
標準アクション	アクションにより自動生成	any	リスト参照以外	いずれかの定義 + <code>action: <action></code> (他エリアに適用されるアクションの場合は + <code>focusable: false</code>)

ドロップダウンリストの使い方

オブジェクトの使用

この機能は 4D プロジェクトでのみ利用可能です。

ドロップダウンリストのデータソースとして、コレクションを内包した オブジェクト を使用できます。このオブジェクトには、次のプロパティが格納されていく必要があります：

なりません:

プロパティ	タイプ	説明
values	コレクション	必須 - スカラー値のコレクション。すべての同じ型の値でなくてはなりません。サポートされている型: <ul style="list-style-type: none">文字列数値日付時間 空、または未定義の場合、ドロップダウンリストは空になります
index	number	選択項目のインデックス (0 と <code>collection.length-1</code> の間の値)。-1 に設定すると、プレースホルダー文字列として <code>currentValue</code> が表示されます。
currentValue	Collection 要素と同じ	選択中の項目 (コードにより設定した場合はプレースホルダーとして使用される)

オブジェクトにその他のプロパティが含まれている場合、それらは無視されます。

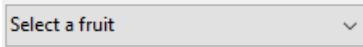
ドロップダウンリストに関連付けるオブジェクトを初期化するには、次の方法があります:

- プロパティリストのデータソーステーマにおいて、選択リストの項目で "<Static List>" を選び、デフォルト値のリストを入力します。これらのデフォルト値は、オブジェクトへと自動的にロードされます。
- オブジェクトとそのプロパティを作成するコードを実行します。たとえば、ドロップダウンリストに紐づいた 変数 が "myList" であれば、On Load フォームイベントに次のように書けます:

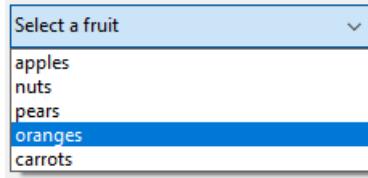
```
// Form.myDrop はフォームオブジェクトのデータソースです

Form.myDrop:=New object
Form.myDrop.values:=New collection("apples"; "nuts"; "pears"; "oranges"; "carrots")
Form.myDrop.index:=-1 // currentValue をプレースホルダーに使います
Form.myDrop.currentValue:="フルーツを選択してください"
```

ドロップダウンリストには、プレースホルダー文字列が表示されます:



ユーザーによって項目が選択されると:



```
Form.myDrop.values // ["apples", "nuts", "pears", "oranges", "carrots"]
Form.myDrop.currentValue // "oranges"
Form.myDrop.index // 3
```

配列の使用

配列 とは、メモリー内の値のリストのことです、配列の名前によって参照されます。ドロップダウンリストをクリックすると、その配列を値のリストとして表示します。

ドロップダウンリストに関連付ける配列を初期化するには、次の方法があります:

- プロパティリストのデータソーステーマにおいて、選択リストの項目で "<Static List>" を選び、デフォルト値のリストを入力します。これらのデフォルト値は、配列へと自動的にロードされます。オブジェクトに関連付けた変数名を使用して、この配列を参照することができます。
- オブジェクトが表示される前に、値を配列要素に代入するコードを実行します。たとえば:

```
ARRAY TEXT(aCities;6)
aCities{1}:="Philadelphia"
aCities{2}:="Pittsburg"
aCities{3}:="Grand Blanc"
aCities{4}:="Bad Axe"
aCities{5}:="Frostbite Falls"
aCities{6}:="Green Bay"
```

この場合、フォームのオブジェクトに紐付けた **変数** は `aCities` でなければなりません。このコードをフォームメソッド内に置き、`On Load` フォームイベント発生時に実行されるようにします。

- オブジェクトが表示される前に、`LIST TO ARRAY` コマンドを使ってリストの値を配列にロードします。たとえば:

```
LIST TO ARRAY("Cities";aCities)
```

この場合にも、フォームのオブジェクトに紐付けた **変数** は `aCities` でなければなりません。このコードは、前述した代入命令文の代わりに実行できます。

ユーザーがおこなった選択内容をフィールドに保存する必要があれば、レコードの登録後に代入命令を実行します。たとえば、次のような Case文のコードを作成します:

```
Case of
:(Form event=On Load)
  LIST TO ARRAY("Cities";aCities)
  If(Record number([People])<0) // 新規レコードの場合
    aCities:=3 // デフォルトの値を表示します
  Else // 既存レコードの場合には、保存された値を表示します
    aCities:=Find in array(aCities;City)
  End if
:(Form event=On Clicked) // ユーザーが選択を変更した場合
  City:=aCities{aCities} // フィールドに新しい値を代入
:(Form event=On Validate)
  City:=aCities{aCities}
:(Form event=On Unload)
  CLEAR VARIABLE(aCities)
End case
```

プロパティリストのイベントテーマにおいて、作成した Case 文の中で使用する各イベントを選択して有効化します。配列には常に有限数の項目が納められます。項目リストは動的であり、メソッドを用いて変更可能です。配列の項目は変更・並び替え・追加することができます。

選択リストの使用

入力エリア（列挙型のフィールドまたは変数）の管理のためにリストボックスを使用したい場合、フィールドまたは変数をドロップダウンリストの **データソース** として直接参照することができます。これにより列挙型のフィールド/変数を容易に管理できるようになります。

たとえば、"White"、"Blue"、"Green"、"Red" という値のみを含む "Color" というフィールドがあった場合、これらの値を含むリストを作成し、それを "Color" フィールドを参照するドロップダウンリストに関連付けることができます。こうすることによって、あとは 4D が自動的にカレント値の入力や表示に関して管理してくれます。

階層リストの場合、第一階層の値のみが表示・選択できます。階層リストの場合、第一階層の値のみが表示・選択できます。

ドロップダウンリストをフィールドや変数と関連付けるには、プロパティリストの **変数あるいは式** 欄にフィールドまたは変数の名前を直接入力します。

この原理は、オブジェクトや配列を用いたドロップダウンリストと組み合わせることはできません。"変数あるいは式" の欄にフィールド名を入力した場合は、必ず選択リストを使用します。

フォームを実行すると、4D が自動的に入力中または表示中のドロップダウンリストの状態を管理します。ユーザーが値を選択すると、その値はフィールドに保存され、このフィールドの値はフォームが表示されたときにドロップダウンリスト表示されます:

Windows



macOS

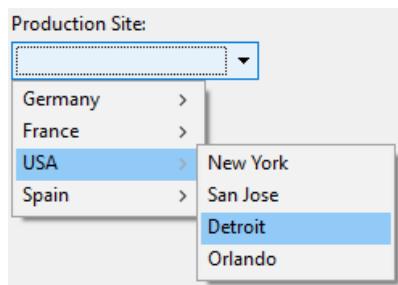


選択された項目値 または 選択された項目参照

選択リスト型のドロップダウンリストをフィールドや変数と関連付けた場合、[データタイプ](#) プロパティを 選択された項目値 または 選択された項目参照 に 設定することができます。このオプションにより、保存するデータのサイズを最適化できるようになります。

階層型選択リストの使用

階層型ドロップダウンリストは、リストの各項目にサブリストが関連付けられています。以下は、階層型ドロップダウンリストの例です：



フォーム上では、階層型ドロップダウンリストは 2階層に制限されています。

階層型選択リストをドロップダウンリストオブジェクトに割り当てるには、プロパティリストの [選択リスト](#) 欄を使います。

階層型ドロップダウンリストの管理には、4Dランゲージの 階層リスト コマンドを使用します。 (*; "name") シンタックスをサポートするすべてのコマンドを、階層型ドロップダウンリストに使用できます (例: [List item parent](#))。

標準アクションの使用

[標準アクション](#) を使って、ドロップダウンリストを自動的に構築することができます。この機能は、以下のコンテキストでサポートされています：

- [gotoPage](#) 標準アクションの使用。この場合、4D は選択された項目の番号に対応する [フォームのページ](#) を自動的に表示します。たとえば、ユーザーが 3番目の項目をクリックすると、4Dはカレントフォームの 3ページ目 (存在する場合) を表示します。実行時のデフォルトでは、ドロップダウンリストにはページ番号 (1, 2...) が表示されます。
- 項目のサブリストを表示する標準アクションの使用 (例: [backgroundColor](#))。この機能には以下の条件があります：
 - スタイル付きテキストエリア ([4D Write Pro エリア](#) または [マルチスタイル](#) プロパティ付き [入力](#)) が標準アクションのターゲットとしてフォーム内に存在する。
 - ドロップダウンリストに [フォーカス可](#) 設定されていない。実行時のドロップダウンリストは、背景色などの値の自動リストを表示します。この自動リストは、各項目が任意の標準アクションを割り当てられた選択リストを設定することで上書きすることもできます。

この機能は、階層型のドロップダウンリストでは使用できません。

プロパティ一覧

[タイプ](#) - [オブジェクト名](#) - [変数あるいは式](#) - [式の型](#) - [値を記憶](#) - [CSSクラス](#) - [ボタンスタイル](#) - [選択リスト](#) - [データタイプ \(式の型\)](#) - [データタイプ \(リスト\)](#) - [左](#) - [上](#) - [右](#) - [下](#) - [幅](#) - [高さ](#) - [横方向サイズ変更](#) - [縦方向サイズ変更](#) - [フォーカス可](#) - [文字フォーマット](#) - [日付フォーマット](#) - [時間フォーマット](#) - [表示状態](#) - [レンダリングしない](#) - [フォント](#) - [フォントサイズ](#) - [太字](#) - [イタリック](#) - [下線](#) - [フォントカラー](#) - [ヘルプTips](#) - [標準アクション](#)

グループボックス

グループボックスはスタティックオブジェクトの一つで、複数のフォームオブジェクトを視覚的にまとめます。

The screenshot shows a light gray rectangular box labeled "Employee Info". Inside, there are three input fields: "First name: [Employee]firstName", "Last name: [Employee]lastName", and "Salary: [Employee]salary". Each field has a placeholder text starting with "[Employee]" followed by the field name.

グループボックスの名前はスタティックテキストです。これは他の 4D ラベルと同様にローカライズすることができます (4D デザインリファレンスの [スタティックテキスト中で参照を使用する](#) および [付録 B: XLIFF アーキテクチャー](#) の章を参照ください)。

JSON 例:

```
"myGroup": {  
    "type": "groupBox",  
    "title": "Employee Info"  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20  
}
```

プロパティ一覧

タイプ - オブジェクト名 - タイトル - CSSクラス - 左 - 上 - 右 - 下 - 幅 - 高さ - 横方向サイズ変更 - 縦方向サイズ変更 - 表示状態 - フォント - フォントサイズ - 太字 - イタリック - 下線 - フォントカラー - 横揃え

入力

入力オブジェクトを使って、データベースフィールド や 変数 といった式をフォーム上に追加することができます。入力オブジェクトは文字ベースのデータ（テキスト、日付、数値など）およびピクチャー型のデータを扱えます。



入力オブジェクトには [代入可、または代入不可の式](#) を設定することができます。

また、入力オブジェクトは [入力可、または不可](#) に設定することができます。入力可に設定された入力オブジェクトはデータを受け入れます。データ入力に対しては、データ入力制御を設定できます。入力不可に設定された入力オブジェクトは、ユーザーの編集を受け付けず、値を表示するのみです。

データは、[オブジェクトメソッドやフォームメソッド](#) を使って管理することができます。

JSON 例：

```
"myText": {  
    "type": "input",      // オブジェクトタイプ  
    "spellcheck": true, // 自動スペルチェック  
    "left": 60,          // フォーム上の座標（左）  
    "top": 160,          // フォーム上の座標（上）  
    "width": 100,         // 幅  
    "height": 20          // 高さ  
}
```

プロパティ一覧

タイプ - オブジェクト名 - 変数あるいは式 - 式の型 - CSSクラス - 選択リスト - 左 - 上 - 右 - 下 - 幅 - 高さ - 横方向サイズ変更 - 縦方向サイズ変更 - 入力可 - 入力フィルター - プレースホルダー - 自動スペルチェック - 複数行 - コンテキストメニュー - 選択を常に表示 - デフォルト値 - 指定リスト - 除外リスト - 文字フォーマット - 数値フォーマット - テキスト (True時)/テキスト (False時) - テキスト (True時)/テキスト (False時) - 日付フォーマット - 時間フォーマット - ピクチャーフォーマット - 表示状態 - ワードラップ フォーカスの四角を隠す - 横スクロールバー - 縦スクロールバー - 塗りカラー - 境界線スタイル - フォント - フォントサイズ - 太字 - イタリック - 下線 - フォントカラー - 方向 - スタイルタグを全て保存 - 横揃え - マルチスタイル - ピッカーの使用を許可 - 印刷時可変 - ドラッグ有効 - ドロップ有効

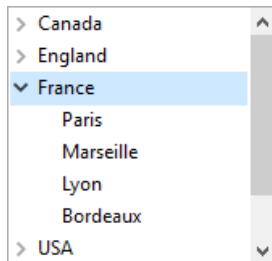
入力の代替手段

フィールドや変数などの式は、フォーム内において入力オブジェクト以外を用いて表示することができます。具体的には以下の方法があります：

- データベースのフィールドから [セレクション型のリストボックス](#) へと、データを直接表示・入力することができます。
- [ポップアップメニュー/ドロップダウンリスト](#) と [コンボボックス](#) オブジェクトを使用することによって、リストフィールドまたは変数をフォーム内にて直接表示することができます。
- ブール型の式は [チェックボックス](#) や [ラジオボタン](#) オブジェクトを用いて提示することができます。

階層リスト

階層リストはフォームオブジェクトの一つで、展開/折りたたみ可能な一つ以上の階層を持つリスト形式でデータを表示するのに使用できます。



展開/折りたたみアイコンは必要に応じて項目の左に自動的に表示されます。階層リストのレベル数に制限はありません。

階層リストのデータソース

階層リストの中身は次の方法で初期化することができます:

- 既存の [選択リスト](#) を関連づけます。選択リストはあらかじめデザインモードにてリストエディターを使って定義します。
- 階層リスト参照を直接 [変数あるいは式](#) に設定します。

ランタイムにおいては、4D ランゲージの [階層リスト](#) コマンドを使って階層リストを管理しますが、その際には対象となるリストの *ListRef* 参照を用います。

ListRef とオブジェクト名

階層リストはメモリ上に存在する ランゲージオブジェクト であると同時に フォームオブジェクト でもあります。

ランゲージオブジェクト は倍長整数型のユニークな内部IDで参照されます (4D ランゲージリファレンスでは *ListRef* と表記)。この ID はリストを作成する `New list`、`Copy list`、`Load list`、`BL0B to list` から返されます。ランゲージオブジェクトのインスタンスはひとつのみしかメモリ中に存在せず、このオブジェクトに対しておこなわれた変更は、これを使用しているすべての場所に即座に反映されます。

フォームオブジェクト はユニークである必要はありません。同じフォームや異なるフォーム上で同一の階層リストを使用することができます。他のフォームオブジェクト同様、ランゲージ中でオブジェクトを指定するにはシンタックス `(*;"ListName")` を使用します。

"ランゲージオブジェクト" としての階層リストと、"フォームオブジェクト" としての階層リストは、*ListRef* の値を格納した中間的な変数により接続されます。たとえば、`mylist` 変数 をフォームオブジェクトに設定した場合には、次のように書けます:

```
mylist:=New list
```

特定のリストを使用している複数のフォームオブジェクトがある場合、それらにはそれぞれに固有の性質と、それらの間で共有される性質を持ちます。以下の性質はリストフォームオブジェクトごとに固有のものです:

- 選択された項目
- 項目の展開/折りたたみ状況
- スクロールカーソルの位置

それ以外の性質 (フォント、フォントサイズ、スタイル、入力制御、カラー、リストの内容、アイコン等) は他のリストフォームオブジェクトと共有され、個別に変更することはできません。したがって、展開/折りたたみ状況に基づくコマンドやカレントの項目に関するコマンド、たとえば `Count list items` を (最後の `*` 引数を渡さずに) 使用するとき、どのフォームオブジェクトに対する処理なのかを明示的に指定することが重要です。

メモリ中の階層リストを指定するには、ランゲージコマンドで `ListRef` IDを使用しなければなりません。フォーム上の階層リストオブジェクトを指定する場合は、コマンド中でシンタックス `(*;"ListName")` を用いてオブジェクト名 (文字列) を使用します。

フォームオブジェクト名に基づくシンタックスをプロパティ設定用のコマンドで使用することは、対象を指定オブジェクトに限定する意味ではなく、むしろ指定オブジェクトの状態に基づいてコマンドが動作することを意味します。複数の階層リスト間で共有されている性質を変更すると、それらすべてに反映されます。たとえば、次のコードを実行すると:

```
SET LIST ITEM FONT(*;"mylist1";*;thefont)
```

mylist1 フォームオブジェクトに関連付けられた階層リスト項目のフォントを変更します。コマンドは *mylist1* オブジェクトの現在選択されている項目を対象としますが、変更はすべてのプロセスのすべてのリストに反映されます。

@をサポート

他のオブジェクトプロパティ管理コマンドのように、`ListName` 引数で “@” 文字を使用できます。このシンタックスは、フォーム上の複数のオブジェクトを指定するために使用されます。しかし階層リストコマンドのコンテキストにおいては、これはすべての場合に適用されるわけではありません。コマンドのタイプにより、このシンタックスは 2つの異なる効果があります：

- プロパティ設定用のコマンドにおいて、このシンタックスは該当する名前のオブジェクトを対象とします（標準の動作）。たとえば、引数 "LH@" は、オブジェクト名が "LH" で始まる階層リストを指定します。
 - `DELETE FROM LIST`
 - `INSERT IN LIST`
 - `SELECT LIST ITEMS BY POSITION`
 - `SET LIST ITEM`
 - `SET LIST ITEM FONT`
 - `SET LIST ITEM ICON`
 - `SET LIST ITEM PARAMETER`
 - `SET LIST ITEM PROPERTIES`
- プロパティ取得用のコマンドにおいて、このシンタックスは該当する名前を持つ最初のオブジェクトを対象とします：
 - `Count list items`
 - `Find in list`
 - `GET LIST ITEM`
 - `Get list item font`
 - `GET LIST ITEM ICON`
 - `GET LIST ITEM PARAMETER`
 - `GET LIST ITEM PROPERTIES`
 - `List item parent`
 - `List item position`
 - `Selected list items`

階層リストに対し利用できる汎用コマンド

いくつかの 4D の汎用コマンドを使用して、フォーム上の階層リストオブジェクトの見た目を変更することができます。これらのコマンドには、* を用いたシンタックスを使用して階層リストのオブジェクト名を渡すか、あるいは標準シンタックスを使用して階層リストの `ListRef` 参照を格納している変数を渡します。

- `OBJECT SET FONT`
- `OBJECT SET FONT STYLE`
- `OBJECT SET FONT SIZE`
- `OBJECT SET COLOR`
- `OBJECT SET FILTER`
- `OBJECT SET ENTERABLE`
- `OBJECT SET SCROLLBAR`
- `OBJECT SET SCROLL POSITION`
- `OBJECT SET RGB COLORS`

注記： `OBJECT SET SCROLL POSITION` コマンドを除き、これらのコマンドでオブジェクト名を指定したとしても、変更は同じリストを使うすべてのオブジェクトに反映されます。

プロパティコマンドの優先順位

階層リストの特定のプロパティ（たとえば 入力可 属性やカラーなど）は、3つの異なる方法で設定することができます：デザインモードのプロパティリスト、"オブジェクトプロパティ" テーマのコマンド、"階層リスト" テーマのコマンド。これら 3つの方法すべてを使ってプロパティを設定した場合、以下の優先順位が適用されます：

1. "階層リスト" テーマのコマンド
2. 汎用のオブジェクトプロパティコマンド
3. プロパティリストのパラメーター

この原則は、コマンドが呼び出された順番に関係なく適用されます。階層リストコマンドを使用して個々に項目のプロパティを変更すると、同等のオブジェクトプロパティコマンドをその後に呼び出したとしても、その項目に対しては効果を持たなくなります。たとえば `SET LIST ITEM PROPERTIES` コマンドを使用して項目のカラーを変更すると、この項目に対して `OBJECT SET COLOR` コマンドは効果を持たなくなります。

位置あるいは参照による項目の管理

階層リストのコンテンツにアクセスするには、通常は位置または参照のいずれかを使用しておこないます。

- 位置を使用する場合には、4D は画面上に表示されているリスト項目の位置に基づいて項目を特定します。つまり、結果は階層項目が展開されているか折りたたまれているかにより異なります。複数のフォームオブジェクトで同一のリストを使用している場合、オブジェクトごとに展開/折りたたみの状態が異なることに注意が必要です。
- 参照を使用する場合には、リスト項目の *itemRef* IDを参照します。これにより、それぞれの項目を階層リスト中での位置や表示状態に関わらず特定できます。

項目参照番号を使用する (*itemRef*)

階層リストのそれぞれの項目は倍長整数型の参照番号 (*itemRef*) を持ります。この値は開発者が使うためのもので、4D は番号を維持するだけです。

警告: どの倍長整数値も参照として使用できますが、0だけは特別な意味を持ちます。このテーマのほとんどのコマンドで、0は最後にリストに追加された項目を指定するのに使用されます。

参照番号を使用するにあたり、いくつかの Tips を紹介します：

1. 項目をユニーク値で識別する必要がない場合 (初心者レベル)

- 最初の例として、アドレスブックで使用するタブシステムを構築するとします。システムは選択されたタブの番号を返すので、それ以上の情報は必要ありません。この場合、項目参照番号について心配する必要はありません。0以外の値を *itemRef* に渡します。なお、アドレスブックシステムの場合、デザインモードで A-Z のリストを定義することもできる点に留意してください。また、プログラムを使えば、レコードがない文字を除いたリストを作成することもできます。
- 2つ目の例は、データベースを利用すると同時に蓄積していくタイプのキーワードリストを考えます。このリストはセッション終了時に `SAVE LIST` や `LIST TO BLOB` コマンドで保存され、セッション開始時に `Load list` や `BLOB to list` コマンドで再度読み込まれます。このリストをフローティングパレットに表示し、ユーザーがキーワードをクリックすると、最前面のプロセスの選択されたエリアに項目テキストが挿入されます。重要なのは、`Selected list items` コマンドは選択項目の位置を返すため、選択された項目のみを扱うということです。この位置情報を `GET LIST ITEM` コマンドに渡せば、項目テキストが取得できます。この例でも、個々の項目を識別する必要がないため、リスト構築の際は *itemRef* 引数に 0以外の任意の数値を渡すことができます。

2. 部分的にリスト項目を識別する必要がある場合 (中級者レベル)

その項目を処理する際に必要となる情報をあらかじめ項目参照番号に格納することができます。この例は `APPEND TO LIST` コマンドの [例題](#) で説明しています。この例題では、項目参照番号にレコード番号を格納しています。また、[Department] レコード由来の項目と [Employees] レコード由来の項目を区別する必要があり、この点も例題にて説明されています。

3. すべての項目リストを個々に識別する必要がある場合 (上級レベル)

リストの全レベルにおいて、個々の項目を識別する必要のある複雑な階層リスト管理プログラムを作成する必要があるとします。これを実装する簡単な方法は独自のカウンターを使用することです。 `APPEND TO LIST` コマンドを使用して *hList* リストを作成するとします。ここで *vhlCounter* 変数を1に初期化します。 `APPEND TO LIST` や `INSERT IN LIST` を呼び出すたびに、このカウンターをインクリメントし (`vhlCounter:=vhlCounter+1`)、カウンター値を項目参照番号に設定します。項目を削除する場合でもカウンターをデクリメントしないことが重要です。つまりカウンターは増え続けるのみです。この方法で、ユニークな項目参照番号を保証できます。番号は倍長整数型なので、20億以上の項目をリストに追加したり挿入したりできます（もっとも、こんなにも多くのデータを扱うのであれば、リストではなくテーブルを使用したほうが良いですが）。

ビットワイス演算子を使用して、項目参照番号に情報を格納することもできます。たとえば 2つの整数値、4バイト値、32個のブール値などです。

どのような場合にユニークな参照番号が必要ですか？

階層リストをユーザーインターフェースとして使用し、クリックまたはドラッグにより選択された項目のみを処理する場合は、ほとんどの場合項目参照番号を必要としません。 `Selected list items` や `GET LIST ITEM` を使用すれば、現在選択されている項目を扱うことができます。さらに `INSERT IN LIST` や `DELETE FROM LIST` などのコマンドは、選択された項目との相対位置でリストを操作できます。

基本的に、項目の選択に関係なく、プログラムで任意のリスト項目にアクセスする必要がある場合に項目参照番号が必要です。

編集可能項目

ユーザーが階層リストの項目を変更できるかどうかを管理することができます。階層リストの項目が修正可能である場合、Alt+クリック (Windows) または Optionキー+クリック (macOS) ショートカットを使用するか、または項目のテキスト上でロングクリックすると、編集できるようになります。

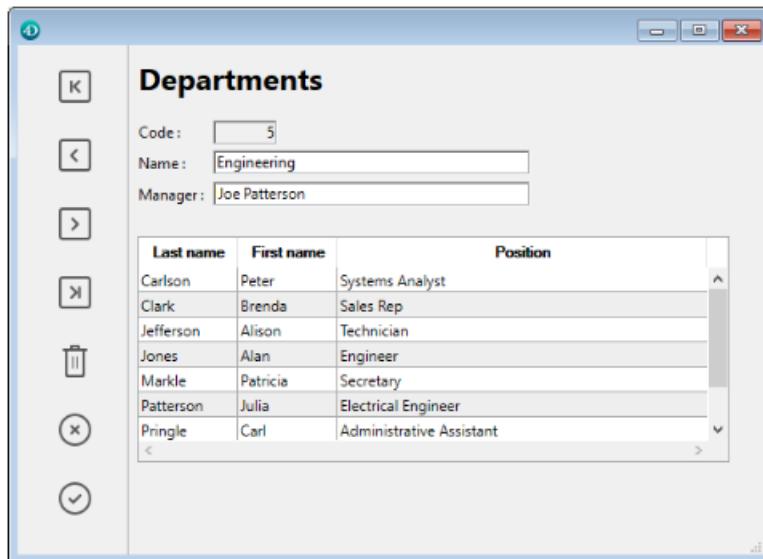
- データソースに関わらず、階層リストフォームオブジェクト全体を [入力可](#) プロパティで管理することができます。
- また、リストエディターで作成したリストを用いて階層リストを生成する場合は、リストエディターの編集可能項目 オプションを使用して、階層リストの項目の修正が可能かどうかを管理することができます。詳細については [リストプロパティの設定](#) を参照してください。

プロパティ一覧

タイプ - オブジェクト名 - 変数あるいは式 - CSSクラス - 選択リスト - 左 - 上 - 右 - 下 - 幅 - 高さ - 横方向サイズ変更 - 縦方向サイズ変更 - 入力可 - フォーカス可 - 入力フィルター - 表示状態 - フォーカスの四角を隠す - 横スクロールバー - 縦スクロールバー - 塗りカラー - 境界線スタイル - フォント - フォントサイズ - 太字 - イタリック - 下線 - フォントカラー - ヘルプTips - ドラッグ有効 - ドロップ有効 - 複数選択可

リストボックス

リストボックスは複合アクティブオブジェクトで、同期化された複数列（カラムとも呼びます）の形式でデータの表示・入力がおこなえます。リストボックスは、エンティティセレクションやレコードセレクションなどのデータベースコンテンツのほか、コレクションや配列などのランゲージコンテンツと紐づけることができます。データ入力、列の並べ替え、イベント管理、外観のカスタマイズ、列の移動など、リストボックスには高度な機能が備わっています。



リストボックスには 1つ以上の列があり、その内容が自動的に同期化されます。理論上、列数に制限はありません（マシンのリソースに依存します）。

概要

基本のユーザー機能

実行中、リストボックスはリストとしてデータを表示し、入力を受け付けます。セルを編集可能にするには（[その列について入力が許可されなければ](#)）、セル上で2回クリックします：

Last name	First name
James	Henry
Jameson	Marc

リストボックスのセルには、複数行のテキストを入力・表示できます。セル内で改行するには、Ctrl+Return (Windows) または Command+Return (macOS) を押します。

セルには布尔やピクチャー、日付、時間、数値も表示することができます。ヘッダーをクリックすると、列の値をソートできます（[標準ソート](#)）。すべての列が自動で同期されます。

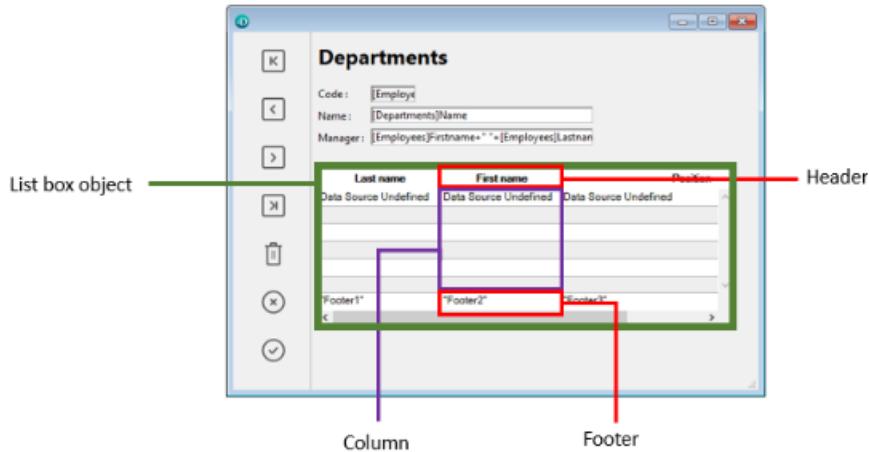
またそれぞれの列幅を変更できるほか、ユーザーはマウスを使用して [列](#) や [行](#) の順番を（そのアクションが許可されていれば）入れ替えることもできます。リストボックスは [階層モード](#) で使用することもできます。

ユーザーは標準のショートカットを使用して 1つ以上の行を選択できます。Shift+クリック で連続した行を、Ctrl+クリック (Windows) や Command+クリック (macOS) で非連続行を選択できます。

リストボックスの構成要素

リストボックスオブジェクトは、以下4つの項目で構成されます：

- リストボックスオブジェクトの全体
- 列
- 列ヘッダー
- 列フッター



それぞれが独自のオブジェクト名や固有のプロパティを持ちます。たとえば、列の数や、交互に使用する行の背景色などはリストボックスオブジェクトのプロパティで指定し、各列の幅は列プロパティ、ヘッダーのフォントはヘッダープロパティで指定します。

リストボックスオブジェクトやリストボックスの各列に対して、オブジェクトメソッドを設定することができます。オブジェクトメソッドの呼び出しは、次の順でおこなわれます：

1. 各列のオブジェクトメソッド
2. リストボックスのオブジェクトメソッド

[ヘッダー](#) と [フッター](#) で発生したイベントは、その列のオブジェクトメソッドが受け取ります。

リストボックスの型

リストボックスには複数のタイプがあり、動作やプロパティの点で異なります。リストボックスの型は[データソースプロパティ](#) で定義します：

- 配列：各列に 4D 配列を割り当てます。配列タイプのリストボックスは [階層リストボックス](#) として表示することができます。
- セレクション（カレントセレクション または 命名セレクション）：各列に式（たとえばフィールド）を割り当てます。それぞれの行はセレクションのレコードを基に評価されます。
- コレクションまたはエンティティセレクション：各列に式を割り当てます。各行の中身はコレクションの要素ごと、あるいはエンティティセレクションのエンティティごとに評価されます。

1つのリストボックス内に複数のデータソースタイプを組み合わせて指定することはできません。データソースは、リストボックス作成時に定義され、プログラムによって後から変更することはできません。

リストボックスの管理

リストボックスオブジェクトはプロパティによってあらかじめ設定可能なほか、プログラムにより動的に管理することもできます。

4D ランゲージにはリストボックス関連のコマンドをまとめた "リストボックス" テーマが専用に設けられていますが、"オブジェクトプロパティ" コマンドや `EDIT ITEM`、`Displayed line number` コマンドなど、ほかのテーマのコマンドも利用することができます。詳細については [4D ランゲージリファレンスマニュアル](#) の[リストボックスコマンド一覧](#)を参照してください。

リストボックスオブジェクト

配列リストボックス

配列リストボックスでは、それぞれの列に 4D の 1次元配列を割り当てなければなりません。ポインター配列を除きすべてのタイプの配列を使用できます。行数は配列の要素数により決定されます。

デフォルトで 4D は各列に "ColumnX" という名前を割り当てます。この配列変数名は [列のプロパティ](#) で変更できます（プロパティリストの [変数あるいは式](#) プロパティを使用します）。列ごとの表示フォーマットを指定するには、`OBJECT SET FORMAT` コマンドも使用できます。

配列タイプのリストボックスは、特別なメカニズムをもつ [階層モード](#) で表示することができます。

配列タイプのリストボックスでは、入力あるいは表示される値は 4D ランゲージで制御します。列に [選択リスト](#) を割り当て、データ入力を制御することも

できます。リストボックスのハイレベルコマンド (`LISTBOX INSERT ROWS` や `LISTBOX DELETE ROWS` 等) や配列操作コマンドを使用して、列の値を管理します。たとえば、列の内容を初期化するには、以下の命令を使用できます:

```
ARRAY TEXT(varCol;size)
```

リストを使用することもできます:

```
LIST TO ARRAY("ListName";varCol)
```

警告: 異なる配列サイズの列がリストボックスに含まれる場合、もっとも小さい配列サイズの数だけを表示します。そのため、各配列の要素数は同じにしなければなりません。リストボックスの列が一つでも空の場合 (ランゲージにより配列が正しく定義またはサイズ設定されなかったときに発生します)、リストボックスは何も表示しません。

セレクションリストボックス

このタイプのリストボックスでは、列ごとにフィールド (例: `[Employees] LastName`) や式を割り当てます。式は 1つ以上のフィールド (たとえば `[Employees] FirstName + " " [Employees] LastName`) または単にフォーミュラ (たとえば `String(Milliseconds)`) を使用できます。式にはプロジェクトメソッド、変数、あるいは配列項目も指定できます。カラムをプログラムで変更するには、`LISTBOX SET COLUMN FORMULA` および `LISTBOX INSERT COLUMN FORMULA` コマンドを使用します。

それぞれの行はセレクションのレコードを基に評価されます。セレクションは カレントセレクション または 命名セレクションです。

データソースがカレントセレクションである場合、データベースに対しておこなわれた変更はリストボックスに自動で反映され、またリストボックスへの変更も自動で データベースに適用されます。つまりカレントセレクションは常に両方で同じです。

コレクションまたはエンティティセレクションリストボックス

このタイプのリストボックスでは、各カラムに式が割り当てられている必要があります。各行の中身はコレクション要素ごと、あるいはエンティティセレクションのエンティティごとに評価されます。

コレクションの各要素、またはエンティティセレクションの各エンティティは、`This` キーワードを用いてオブジェクトとして取得します。カラムの式にはプロパティパス、プロジェクトメソッド、変数、あるいはフォーミュラが指定可能で、`This` を通して得た各エンティティあるいはコレクション要素オブジェクトが利用できます。例: `This.<propertyPath>` (あるいはスカラー値のコレクションの場合は `This.value`)。カラムをプログラムで変更するには、`LISTBOX SET COLUMN FORMULA` および `LISTBOX INSERT COLUMN FORMULA` コマンドを使用します。

データソースがエンティティセレクションの場合、リストボックス側に対しておこなった変更は自動的にデータベースに保存されます。その一方で、データベース側に対しておこなった変更は、該当エンティティがリロードされてはじめてリストボックス側に反映されます。

データソースがコレクションの場合、リストボックス内の値に変更をおこなった場合、その変更はコレクションにも反映されます。その一方で、コレクションに対して、たとえば [Collection クラス](#) の様々な関数を使用して変更をおこなった場合、コレクション変数を自らに再代入することで明示的に 4D に通知する必要があり、それによってリストボックスのコンテンツは更新されます。たとえば:

```
myCol:=myCol.push("new value") // リストボックスに new value を表示
```

プロパティ一覧

提供されるプロパティはリストボックスのタイプに依存します。

プロパティ	配列リストボックス	セレクションリストボックス	コレクションまたはエンティティセレクションリストボックス
交互に使用する背景色	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
背景色	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
太字	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
背景色式		<input type="radio"/>	<input type="radio"/>
境界線スタイル	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

プロパティ	配列リストボックス	セレクションリストボックス	コレクションまたはエンティティセレクションリストボックス
クラス	○	○	○
コレクションまたはエンティティセレクション		○	○
カラム自動リサイズ	○	○	○
カレントの項目			○
カレントの項目の位置			○
データソース	○	○	○
詳細フォーム名		○	
ヘッダーを表示	○	○	○
フッターを表示	○	○	○
行をダブルクリック		○	
ドラッグ有効	○	○	○
ドロップ有効	○	○	○
フォーカス可	○	○	○
フォント	○	○	○
フォントカラー	○	○	○
フォントカラー式		○	○
フォントサイズ	○	○	○
高さ (リストボックス)	○	○	○
高さ (ヘッダー)	○	○	○
高さ (フッター)	○	○	○
追加の空白の行を非表示	○	○	○
フォーカスの四角を隠す	○	○	○
セレクションハイライトを非表示	○	○	○
階層リストボックス	○		
ハイライトセット		○	
横揃え	○	○	○
横線カラー	○	○	○
横スクロールバー	○	○	○
横方向サイズ変更	○	○	○
イタリック	○	○	○
左	○	○	○
マスターーテーブル		○	
メタ情報式			○
メソッド	○	○	○
行の移動可	○		
命名セレクション		○	
列数	○	○	○
スクロールしない列数	○	○	○

プロパティ	配列リストボックス	セレクションリストボックス	コレクションまたはエンティティコレクションリストボックス
オブジェクト名	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
右	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
行背景色配列	<input type="radio"/>		
行コントロール配列	<input type="radio"/>		
行フォントカラー配列	<input type="radio"/>		
行の高さ	<input type="radio"/>		
行高さ配列	<input type="radio"/>		
行スタイル配列	<input type="radio"/>		
選択された項目			<input type="radio"/>
選択モード	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
シングルクリック編集	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
ソート可	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
標準アクション	<input type="radio"/>		
スタイル式		<input type="radio"/>	<input type="radio"/>
上	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
透過	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
タイプ	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
下線	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
変数あるいは式	<input type="radio"/>	<input type="radio"/>	
縦揃え	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
縦線カラー	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
縦スクロールバー	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
縦方向サイズ変更	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
表示状態	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
幅	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

リストボックスの列、ヘッダーおよびフッターにもそれぞれ固有のプロパティがあります。

フォームイベント

フォームイベント	取得される追加プロパティ（メインプロパティについては Form event 参照）	コメント
On After Edit	<ul style="list-style-type: none"> ● <code>column</code> ● <code>columnName</code> ● <code>row</code> 	
On After Keystroke	<ul style="list-style-type: none"> ● <code>column</code> ● <code>columnName</code> ● <code>row</code> 	
On After Sort	<ul style="list-style-type: none"> ● <code>column</code> ● <code>columnName</code> ● <code>headerName</code> 	複合フォーミュラはソート不可 (例: <code>This.firstName + This.lastName</code>)

イベント	取得される追加プロパティ (emainプロパティについては Form ● columnName)	コメント
On Alternative	● column	配列リストボックスのみ
Click	● columnName ● row	
On Before Data Entry	● column ● columnName ● row	
On Before Keystroke	● column ● columnName ● row	
On Begin Drag Over	● column ● columnName ● row	
On Clicked	● column ● columnName ● row	
On Close Detail	● row	カレントセレクション&命名セレクションリストボックスのみ
On Collapse	● column ● columnName ● row	階層リストボックスのみ
On Column Moved	● columnName ● newPosition ● oldPosition	
On Column Resize	● column ● columnName ● newSize ● oldSize	
On Data Change	● column ● columnName ● row	
On Delete Action	● row	
On Display Detail	● isRowSelected ● row	
On Double Clicked	● column ● columnName ● row	
On Drag Over	● area ● areaName ● column ● columnName ● row	

On Drop	取得される追加プロパティ（emainプロパティについては Form event 参照 ● <code>column</code> ● <code>columnName</code>	コメント
	● <code>row</code>	
On Expand	● <code>column</code> ● <code>columnName</code> ● <code>row</code>	階層リストボックスのみ
On Footer Click	● <code>column</code> ● <code>columnName</code> ● <code>footerName</code>	配列、カレントセレクション&命名セレクションリストボックスのみ
On Getting Focus	● <code>column</code> ● <code>columnName</code> ● <code>row</code>	追加プロパティの取得はセル編集時のみ
On Header Click	● <code>column</code> ● <code>columnName</code> ● <code>headerName</code>	
On Load		
On Losing Focus	● <code>column</code> ● <code>columnName</code> ● <code>row</code>	追加プロパティの取得はセル編集完了時のみ
On Mouse Enter	● <code>area</code> ● <code>areaName</code> ● <code>column</code> ● <code>columnName</code> ● <code>row</code>	
On Mouse Leave		
On Mouse Move	● <code>area</code> ● <code>areaName</code> ● <code>column</code> ● <code>columnName</code> ● <code>row</code>	
On Open Detail	● <code>row</code>	カレントセレクション&命名セレクションリストボックスのみ
On Row Moved	● <code>newPosition</code> ● <code>oldPosition</code>	配列リストボックスのみ
On Selection Change		
On Scroll	● <code>horizontalScroll</code> ● <code>verticalScroll</code>	
On Unload		

追加プロパティ

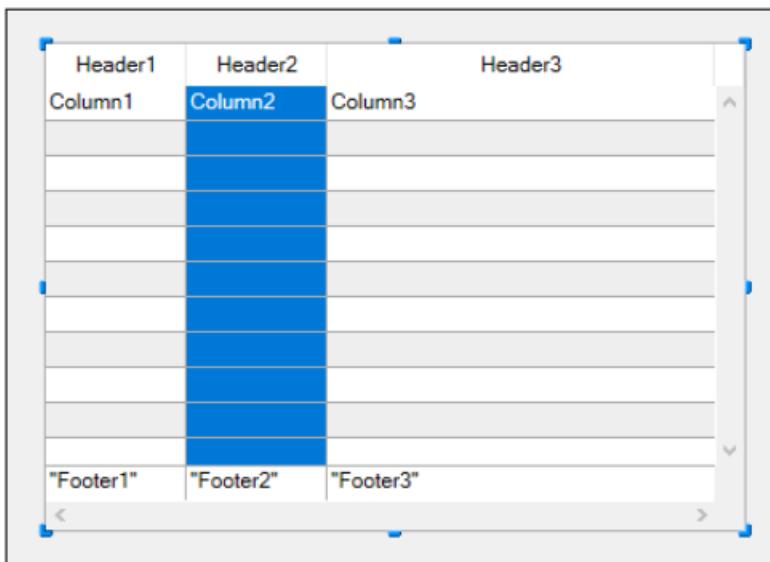
リストボックスやリストボックス列オブジェクトにて発生するフォームイベントは、次の追加プロパティを返すことがあります:

プロパティ	タイプ	説明
area	テキスト	リストボックスオブジェクトエリア ("header", "footer", "cell")
areaName	テキスト	エリアの名称
column	倍長整数	列番号
columnName	テキスト	列の名称
footerName	テキスト	フッターの名称
headerName	テキスト	ヘッダーの名称
horizontalScroll	倍長整数	右方向スクロールの場合は正の数値、左方向の場合は負の数値
isRowSelected	boolean	行が選択されていれば true、でなければ false
newPosition	倍長整数	列あるいは行の変更後の位置
newSize	倍長整数	列または行の変更後のサイズ (ピクセル単位)
oldPosition	倍長整数	列あるいは行の変更前の位置
oldSize	倍長整数	列または行の変更前のサイズ (ピクセル単位)
row	倍長整数	行番号
verticalScroll	倍長整数	下方向スクロールの場合は正の数値、上方向の場合は負の数値

"偽" カラムや存在しないカラムにてイベントが発生した場合には、主に空の文字列が返されます。

リストボックス列

リストボックスは、それぞれ固有のプロパティを持つ 1つ以上の列オブジェクトから構成されています。列を選択するには、フォームエディターでリストボックスオブジェクトが選択されているときに任意の列をクリックします:



リストボックスの各列毎に標準のプロパティ (テキスト、背景色など) を設定できます。設定すると、リストボックスに対する設定よりもこちらが優先されます。

配列型リストボックスのカラムについては、[式タイプ](#) (テキスト、数値、整数、ブール、ピクチャ、時間、日付、あるいはオブジェクト) を定義することができます。

列特有のプロパティ

[オブジェクト名](#) - [変数あるいは式](#) - [式タイプ \(配列リストボックス列\)](#) - [CSSクラス](#) - [デフォルト値](#) - [選択リスト](#) - [式](#) - [データタイプ \(セレクションおよびコレクションリストボックス列\)](#) - [関連付け](#) - [幅](#) - [自動行高](#) - [最小幅](#) - [最大幅](#) - [サイズ変更可](#) - [入力可](#) - [入力フィルター](#) - [指定リスト](#) - [除外リスト](#)

ト - 表示タイプ - 文字フォーマット - 数値フォーマット - テキスト (True時)/テキスト (False時) - 日付フォーマット - 時間フォーマット - ピクチャーフォーマット - 非表示 - ワードラップ エリプシスを使用して省略 - 背景色 - 交互に使用する背景色 - 行背景色配列 - 背景色式 - フォント - 太字 - イタリック - 下線 - 行スタイル配列 - スタイル式 - フォントカラー - 行フォントカラー配列 - 行フォントカラー式 - 横揃え - 縦揃え - マルチスタイル - メソッド

フォームイベント

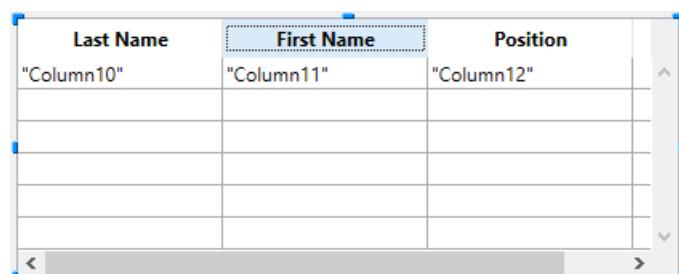
フォームイベント	取得される追加プロパティ (メインプロパティについては Form event 参照)	コメント
On After Edit	<ul style="list-style-type: none"> ● column ● columnName ● row 	
On After Keystroke	<ul style="list-style-type: none"> ● column ● columnName ● row 	
On After Sort	<ul style="list-style-type: none"> ● column ● columnName ● headerName 	複合フォーミュラはソート不可 (例: <code>This.firstName + This.lastName</code>)
On Alternative Click	<ul style="list-style-type: none"> ● column ● columnName ● row 	配列リストボックスのみ
On Before Data Entry	<ul style="list-style-type: none"> ● column ● columnName ● row 	
On Before Keystroke	<ul style="list-style-type: none"> ● column ● columnName ● row 	
On Begin Drag Over	<ul style="list-style-type: none"> ● column ● columnName ● row 	
On Clicked	<ul style="list-style-type: none"> ● column ● columnName ● row 	
On Column Moved	<ul style="list-style-type: none"> ● columnName ● newPosition ● oldPosition 	
On Column Resize	<ul style="list-style-type: none"> ● column ● columnName ● newSize ● oldSize 	
On Data Change	<ul style="list-style-type: none"> ● column ● columnName ● row 	
On Double	<ul style="list-style-type: none"> ● column 	

Clicked フォームイベント	<ul style="list-style-type: none"> ● columnName 取得される追加プロパティ（emainプロパティについてはForm ● row event 参照） 	コメント
On Drag Over	<ul style="list-style-type: none"> ● area ● areaName ● column ● columnName ● row 	
On Drop	<ul style="list-style-type: none"> ● column ● columnName ● row 	
On Footer Click	<ul style="list-style-type: none"> ● column ● columnName ● footerName 	配列、カレントセレクション&命名セレクションリストボックスのみ
On Getting Focus	<ul style="list-style-type: none"> ● column ● columnName ● row 	追加プロパティの取得はセル編集時のみ
On Header Click	<ul style="list-style-type: none"> ● column ● columnName ● headerName 	
On Load		
On Losing Focus	<ul style="list-style-type: none"> ● column ● columnName ● row 	追加プロパティの取得はセル編集完了時のみ
On Row Moved	<ul style="list-style-type: none"> ● newPosition ● oldPosition 	配列リストボックスのみ
On Scroll	<ul style="list-style-type: none"> ● horizontalScroll ● verticalScroll 	
On Unload		

リストボックスヘッダー

リストボックスのヘッダープロパティにアクセスするためには、リストボックスのプロパティリストで[ヘッダーを表示](#) オプションが選択されていなければなりません。

ヘッダーが表示されていれば、フォームエディターでリストボックスオブジェクトが選択されているときに、リストボックスヘッダーをクリックするとヘッダーを選択できます：



リストボックスの各列ヘッダー毎に標準のテキストプロパティを設定できます。設定すると、リストボックスや列に対する設定よりもこちらが優先されます。

さらに、ヘッダー特有のプロパティを設定することができます。カスタマイズされた並び替えなどの用途に、ヘッダーの列タイトルの隣、あるいはタイトルの代わりにアイコンを表示することができます。

 Name	Position
Smith	Administrator

ランタイムにおいてヘッダーで発生したイベントは、[その列のオブジェクトメソッド](#)が受け取ります。

ヘッダーに `OBJECT SET VISIBLE` コマンドを使用すると、このコマンドに渡した引数に関わらず、そのリストボックスのすべてのヘッダーが対象になります。たとえば、`OBJECT SET VISIBLE(*;"header3";False)` という命令の場合、指定したヘッダーだけではなく、`header3` が属するリストボックスの全ヘッダーを非表示にします。

ヘッダー特有のプロパティ

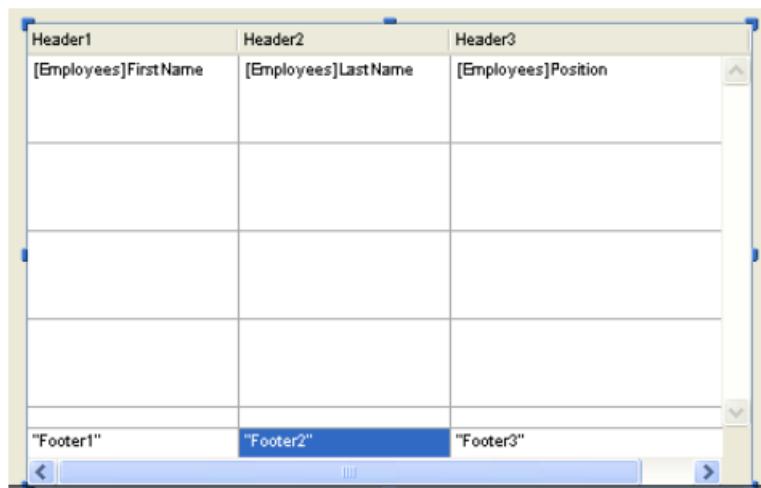
オブジェクト名 - 変数あるいは式 - タイトル - CSSクラス - パス名 - アイコンの場所 - 幅 - フォント - フォントサイズ - 太字 - イタリック - 下線 - フォントカラー - 横揃え - 縦揃え - ヘルプTips

リストボックスフッター

リストボックスのフッタープロパティにアクセスするためには、リストボックスのプロパティリストで [フッターを表示](#) オプションが選択されていなければなりません。

リストボックスは、追加の情報を表示するための入力を受け付けない "フッター" を持つことができます。表形式で表示されるデータについて、合計や平均などの計算値を表示するためにフッターは通常使用されます。

フッターが表示されていれば、フォームエディターでリストボックスオブジェクトが選択されているときにフッターをクリックすることで選択できます：



リストボックスの各列フッター毎に標準のテキストプロパティを設定できます。設定すると、リストボックスや列に対する設定よりもこちらが優先されます。さらに、フッター特有のプロパティを設定することができます。カスタムまたは自動計算をフッターに挿入することができます。

ランタイムにおいてフッターで発生したイベントは、[その列のオブジェクトメソッド](#)が受け取ります。

フッターに `OBJECT SET VISIBLE` コマンドを使用すると、このコマンドに渡した引数に関わらず、そのリストボックスのすべてのフッターが対象になります。たとえば、`OBJECT SET VISIBLE(*;"footer3";False)` という命令の場合、指定したフッターだけではなく、`footer3` が属するリストボックスの全フッターを非表示にします。

フッター特有のプロパティ

オブジェクト名 - 変数あるいは式 - 式の型 - 変数の計算 - CSSクラス - 幅 - 文字フォーマット - 数値フォーマット - 曜日フォーマット - 時間フォーマット - ピクチャーフォーマット - ワードラップ エリプシスを使用して省略 - 背景色 - フォント - フォントサイズ - 太字 - イタリック - 下線 - フォントカラー - 横揃え - 縦揃え - ヘルプTips

入力の管理

リストボックスのセルが入力可能であるには、以下の条件を満たす必要があります:

- セルが属する列が **入力可** に設定されている（でなければ、その列のセルには入力できません）。
- On Before Data Entry** イベントで \$0 が -1 を返さない。カーソルがセルに入ると、その列のメソッドで **On Before Data Entry** イベントが生成されます。このイベントのコンテキストにおいて、\$0 に -1 を設定すると、そのセルは入力不可として扱われます。Tab や Shift+Tab が押された後にイベントが生成された場合には、フォーカスはそれぞれ次あるいは前のセルに移動します。\$0 が -1 でなければ（デフォルトは 0）、列は入力可であり編集モードに移行します。

2つの配列で構築されるリストボックスを考えてみましょう。1つは日付でもう1つはテキストです。日付配列は入力不可ですが、テキスト配列は日付が過去でない場合に入力可とします。

Header1	Header2
Variable Name: tDate	Variable Name: tText

arrText 列のメソッドは以下の通りです:

```
Case of
  :(FORM event.code=On Before Data Entry) // セルがフォーカスを得たとき
    LISTBOX GET CELL POSITION(*;"lb";$col;$row)
  // セルの特定
  If(arrDate{$row}<Current date) // 過去の日付なら
    $0:=-1 // セルは入力不可
  Else
    // そうでなければ入力可
  End if
End case
```

On Before Data Entry イベントは **On Getting Focus** より前に生成されます。

データの整合性を保つため、セレクション型とエンティティセレクション型のリストボックスにおいては、レコード/エンティティに対する変更はセル内の編集が確定されたときに自動的に保存されます。確定は、以下のような場合を指します:

- セルがアクティブでなくなったとき（ユーザーによるタブキー押下、クリック操作など）
- リストボックスからフォーカスが外れたとき
- フォームからフォーカスが外れたとき

データ入力・編集操作にともなって発生するイベントのシーケンスは次のようになります:

動作	リストボック ス型	イベントシーケンス
セルが編集モードに切り替 わったとき (ユーザー操作または <code>EDIT ITEM</code> コマンド)	すべて	On Before Data Entry
	すべて	On Getting Focus
セルの値が編集されたとき	すべて	On Before Keystroke
	すべて	On After Keystroke
	すべて	On After Edit
ユーザーがセルを確定し、セル を移動したとき	セレクショ ンリストボッ クス	保存
	レコードセ レクションリ ストボックス	On saving an existing record トリガー (設定されていれば)
	セレクショ ンリストボッ クス	On Data Change(*)
	エンティティ セレクショ ンリストボッ クス	エンティティはオートマージオプション、オプティミスティック・ロックモードで保存されます (<code>entity.save()</code> を参照ください)。正常に保存できた場合には、エンティティは更新され最新の 状態が表示されます。保存処理が失敗した場合、エラーが表示されます。
	すべて	On Losing Focus

(*) エンティティセレクションリストボックスでの `On Data Change` イベントの場合:

- `カレントの項目` オブジェクトには編集前の値が格納されます。
- `This` オブジェクトには、編集後の値が格納されます。

コレクション/エンティティセレクション型では、式が `null` に評価される場合にリストボックスでのデータ入力に制約があります。この場合、セル内の `null` 値を編集・削除することはできません。

選択行の管理

選択行の管理は、リストボックスのタイプが配列か、レコードのセレクションか、あるいはコレクション/エンティティセレクションかによって異なります。

- セレクションリストボックス: 選択行は、デフォルトで `$ListboxSetX` と呼ばれる変更可能なセットにより管理されます (X は 0 から始まり、フォーム内のリストボックスの数に応じて一つずつ増加していきます)。このセットはリストボックスのプロパティ `リストボックス` で定義します。このセットは 4D が自動で管理します。ユーザーがリストボックス内で 1つ以上の行を選択すると、セットが即座に更新されます。他方、リストボックスの選択をプログラムから更新するために、"セット" テーマのコマンドを使用することができます。
- コレクション/エンティティセレクションリストボックス: 選択項目は、専用のリストボックスプロパティを通して管理されます。
 - `カレントの項目` は、選択された要素/エンティティを受け取るオブジェクトです。
 - `選択された項目` は、選択された項目のコレクションです。
 - `カレントの項目の位置` は、選択された要素あるいはエンティティの位置を返します。
- 配列リストボックス: `LISTBOX SELECT ROW` コマンドを使用して、プログラムからリストボックスの行を選択できます。`リストボックスオブジェクトにリンクされた変数` は、行選択の取得、設定、保存に使用します。この変数は布尔配列で、4Dが自動的に作成・管理します。この配列のサイズは、リストボックスのサイズにより決定されます。つまり、各列に関連付けられた配列のうち、最も小さな配列と同じ数の要素を持ちます。この配列の各要素には、対応する行が選択された場合には `true` が、それ以外の場合は `false` が設定されます。4D は、ユーザーの動作に応じてこの配列の内容を更新します。これとは逆に、この配列要素の値を変更して、リストボックス中の選択行を変更することができます。他方、この配列への要素の挿入や削除はできず、行のタイプ変更もできません。`Count in array` コマンドを使用して、選択された行の数を調べることができます。た

とえば、以下のメソッドは配列タイプのリストボックスで、最初の行の選択を切り替えます：

```
ARRAY BOOLEAN(tListBox;10)
// tListBox はフォーム内にあるリストボックス変数の名前です
If(tListBox{1}:=True)
  tListBox{1}:=False
Else
  tListBox{1}:=True
End if
```

OBJECT SET SCROLL POSITION コマンドは、最初に選択された行または指定された行を表示するようにリストボックスをスクロールします。

選択行の見た目のカスタマイズ

リストボックスの [セレクションハイライトを非表示](#) プロパティにチェックを入れている場合には、他のインターフェースオプションを活用してリストボックスの選択行を可視化する必要があります。ハイライトが非表示になっていても選択行は引き続き 4D によって管理されています。つまり：

- 配列タイプのリストボックスの場合、当該リストボックスにリンクしている布尔配列変数から選択行を割り出します。
- セレクションタイプのリストボックスの場合、特定行（レコード）がリストボックスの [ハイライトセット](#) プロパティで指定しているセットに含まれているかを調べます。

特定された選択行は、それらの背景色やフォントカラー、フォントスタイルなどをプログラムによって調整することで、選択行を独自の方法で可視化することができます。リストボックスのタイプによって、表示の管理は配列や式を使用しておこないます（後述参照）。

リストボックスの現アピアランス（フォントカラー、背景色、フォントスタイル等）を使うには `lk inherited` 定数が使用できます。

セレクションリストボックス

選択行を特定するには、リストボックスの [ハイライトセット](#) プロパティで指定されているセットに対象行が含まれているかを調べます：選択行のアピアランスを定義するには、プロパティリストにて [カラー式またはスタイル式プロパティ](#) を 1つ以上使います。

次の場合には式が自動的に再評価されることに留意ください：

- リストボックスのセレクションが変わった場合
- リストボックスがフォーカスを得た、あるいは失った場合
- リストボックスが設置されたフォームウィンドウが最前面になった、あるいは最前面ではなくなりた場合

配列リストボックス

選択行を特定するには、当該リストボックスにリンクしている布尔配列 [変数](#) を調べます：

選択行のアピアランスを定義するには、プロパティリストにて [行カラー配列または行スタイル配列プロパティ](#) を 1つ以上使います。

選択行のアピアランスを定義するリストボックス配列は、`On Selection Change` フォームイベント内で再計算する必要があることに留意が必要です。また、フォーカスの有無を選択行の表示に反映させるには、次のフォームイベント内でもこれらの配列を変更することができます：

- `On Getting Focus` (リストボックスプロパティ)
- `On Losing Focus` (リストボックスプロパティ)
- `On Activate` (フォームプロパティ)
- `On Deactivate` (フォームプロパティ)

例題

システムのハイライトを非表示にして、リストボックスの選択行を緑の背景色で表しました：

Category	ID	Reference	Value
Alpha	215	5A0DF64-EC5-955F7EA-BD284E4-8A	15,425 ▲
Bravo	196	D9E3484-547-AEECB8B-B1808FF-A6	4,592
Alpha	205	3295824-3A8-B48B870-2074C57-B9	-3,672
Charlie	197	B3800C4-C64-A6C95CB-ED27729-B5	16,212
Echo	214	835F344-8EE-B422E66-5C52074-01	-12,332
Alpha	200	46784B4-B20-AE2E51D-0159EA4-D0	1,283
Delta	213	11F5FD4-E48-9BD6E93-E8B9F82-59	13,236
Delta	203	3E80494-879-9F2CEC2-4008AA4-F5	-12,231
Charlie	202	015D694-9C8-91113AA-B8A51A1-52	27,100
Bravo	211	E998AC4-FAE-93BE025-E4CA634-E8	2,630
Charlie	207	B19F244-A30-A03B668-C407B43-D4	16,677
Delta	208	41B1DE4-D29-BC8E7BF-5062D92-B7	-14,759
Echo	199	7005654-722-926CDCE-D8E1BBD-83	23,952
Delta	198	0AD0734-0C7-BA8168E-A0AB67A-1A	-19,758
Alpha	210	6F46794-0D6-AF0E61A-D43231E-3E	24,342
Bravo	201	00A8334-B4B-8419285-8772CFB-B4	-3,657
Charlie	212	9EF2FD4-B1B-97138DE-B3750BA-FB	-4,850
Echo	209	FD05424-365-B8DB0C2-91098A8-80	2,941
Echo	204	7473724-2FA-82F49A5-010BDDED-98	22,200
Bravo	206	3537D34-A9A-AE41C4E-34EC5B6-43	1,205

配列タイプのリストボックスの場合、[行背景色配列](#) をプログラムにより更新する必要があります。JSON フォームにおいて、リストボックスに次の行背景色配列を定義した場合:

```
"rowFillSource": "_ListboxBackground",
```

リストボックスのオブジェクトメソッドに次のように書けます:

```
Case of
:(FORM event.code=On Selection Change)
$n:=Size of array(LB_Arrays)
ARRAY LONGINT(_ListboxBackground;$n) // 行背景色配列
For($i;1;$n)
  If(LB_Arrays{$i}=True) // 選択されていれば
    _ListboxBackground{$i}:=0x0080C080 // 背景色を緑にします
  Else // 選択されていなければ
    _ListboxBackground{$i}:=lk inherited
  End if
End for
End case
```

セレクションタイプのリストボックスで同じ効果を得るには、[ハイライトセット](#) プロパティで指定されたセットに応じて [背景色式](#) が更新されるよう、メソッドを利用します。

JSON フォームにおいて、リストボックスに次のハイライトセットおよび背景色式を定義した場合:

```
"highlightSet": "$SampleSet",
"rowFillSource": "UI_SetColor",
```

UI_SetColor メソッドに次のように書けます:

```

If(Is in set("$SampleSet"))
    $color:=0x0080C080 // 背景色を緑にします
Else
    $color:=lk inherited
End if

$0:=$color

```

階層リストボックスにおいては、[セレクションハイライトを非表示](#) オプションをチェックした場合には、ブレーク行をハイライトすることができません。同階層のヘッダーの色は個別指定することができないため、任意のブレーク行だけをプログラムでハイライト表示する方法はありません。

ソートの管理

リストボックスには、標準ソートとカスタムソートがあります。リストボックスの特定の列がソートされているとき、他の列も常に自動で同期されます。

標準ソート

ヘッダーがクリックされると、リストボックスはデフォルトで標準ソートによる並べ替えをおこないます。標準的な並べ替えとは、列の評価値を英数字順に並べ替え、続けてクリックされると昇順/降順を交互に切り替えます。

リストボックスの [ソート可](#) プロパティを無効化すると、ユーザーによる標準ソートを禁止することができます（デフォルトは有効）。

標準ソートのサポートは、リストボックスのタイプに依存します：

リストボックスタイプ	標準ソートのサポート	コメント
Object の Collection	○	<ul style="list-style-type: none"> "This.a" や "This.a.b" 列はソート可能です。 リストボックス列の式プロパティ は 代入可能な式 でなくてはなりません。
スカラー値のコレクション	×	orderBy() 関数を使ったカスタムソートを使用します。
エンティティセレクション	○	<ul style="list-style-type: none"> リストボックス列の式プロパティ は 代入可能な式 でなくてはなりません。 ソート可：オブジェクト属性プロパティのソート（例："data" がオブジェクト属性の場合の "This.data.city"） ソート可：リレート属性のソート（例："This.company.name"） ソート不可：リレート属性を介したオブジェクト属性プロパティのソート（例："This.company.data.city"）。この場合には、orderByFormula() 関数を使ったカスタムソートを使用します（後述の例題参照）
カレントセレクション	○	単純な式のみソート可能です（例： <code>[Table_1]Field_2</code> ）
命名セレクション	×	
配列	○	ピクチャーアイコンやポインター配列と紐づけられた列はソートできません

カスタムソート

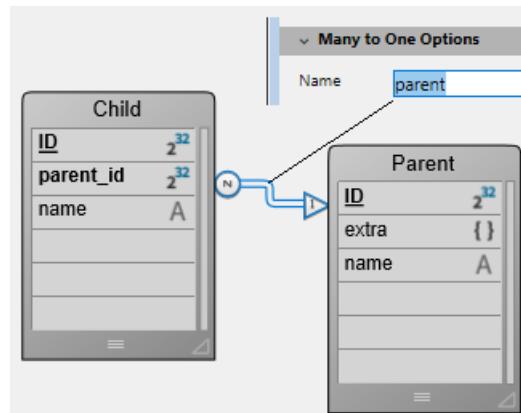
開発者は、[LISTBOX SORT COLUMNS](#) コマンドを使用したり、[On Header Click](#) と [On After Sort](#) フォームイベントを 4Dコマンドと組み合わせて、独自のソートを設定することができます。

カスタムソートを以下のが可能です：

- [LISTBOX SORT COLUMNS](#) コマンドを使って、複数カラムを対象に複数レベルのソートをおこなう
- [collection.orderByFormula\(\)](#) や [entitySelection.orderByFormula\(\)](#) などの関数を使って、複雑な条件のソートをおこなう

例題

リレート先のオブジェクト属性のプロパティ値に基づいてリストボックスをソートします。以下のようなストラクチャーの場合を考えます:



`Form.child` 式に紐づいた、エンティティセレクションタイプのリストボックスを設置します。 `On Load` フォームイベントでは、`Form.child:=ds.Child.all()` を実行します。

次の 2つの列を表示します:

子供の名前	親のニックネーム
<code>This.name</code>	<code>This.parent.extra.nickname</code>

2列目の値に基づいてリストボックスをソートするには、次のコードを書きます:

```
If (Form event code=On Header Click)
    Form.child:=Form.child.orderByFormula("This.parent.extra.nickname"; dk ascending)
End if
```

列ヘッダー変数

列ヘッダー変数の値を使用すると、列の現在の並べ替え状況（読み込み）や並べ替え矢印の表示など、追加情報を管理することができます。

- 変数が 0 のとき、列は並べ替えられておらず、矢印は表示されていません。
- 変数が 1 のとき、列は昇順で並べ替えられており、並べ替え矢印が表示されています。
- 変数が 2 のとき、列は降順で並べ替えられており、並べ替え矢印が表示されています。

列ヘッダー変数には、宣言された、あるいは動的な **変数** のみを使用できます。その他の **式**（例: `Form.sortValue`）はサポートされていません。

変数の値を設定して（たとえば `Header2:=2`）、ソートを表す矢印の表示を強制することができます。しかし、列のソート順は変更されません、これを処理するのは開発者の役割です。

OBJECT SET FORMAT コマンドは、カスタマイズされた並べ替えアイコンをサポートする機能をリストボックスヘッダー用に提供しています。

スタイルとカラー、表示の管理

リストボックスの背景色、フォントカラー、そしてフォントスタイルを設定するためにはいくつかの方法があります:

- リストボックスオブジェクトのプロパティリストを使用
- 列のプロパティリストを使用

- リストボックスまたは列ごとの [配列や式](#) プロパティを使用
- セルごとのテキストにて定義 ([マルチスタイルテキスト](#) の場合)

優先順位と継承

優先順位や継承の原理は、複数のレベルにわたって同じプロパティに異なる値が指定された場合に適用されます。

優先度	設定場所
優先度高	セル単位 (マルチスタイル使用時)
	列の配列/メソッド
	リストボックスの配列/メソッド
	列のプロパティ
	リストボックスのプロパティ
優先度低	メタ情報式 (コレクションまたはエンティティセレクションリストボックスの場合)

例として、リストボックスのプロパティにてフォントスタイルを設定しながら、列には行スタイル配列を使用して異なるスタイルを設定した場合、後者が有効となります。

それぞれの属性 (スタイル、カラー、背景色) について、デフォルトの値を使用した場合、属性の 継承 がおこなわれます:

- セル属性について: 行の属性値を受け継ぎます
- 行属性について: 列の属性値を受け継ぎます
- 列属性について: リストボックスの属性値を受け継ぎます

このように、高次のレベルの属性値をオブジェクトに継承させたい場合は、定義するコマンドに `lk inherited` 定数 (デフォルト値) を渡すか、対応する行スタイル/カラー配列の要素に直接渡します。以下のような、標準のフォントスタイルで行の背景色が交互に変わる配列リストボックスを考えます:

standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard

以下の変更を加えます:

- リストボックスオブジェクトの [行背景色配列](#) プロパティを使用して、2行目の背景色を赤に変更します。
- リストボックスオブジェクトの [行スタイル配列](#) を使用して、4 行目のスタイルをイタリックに変更します。
- 5 列目の列オブジェクトの [行スタイル配列](#) を使用して、5 列目の二つの要素を太字に変更します。
- 1、2 列目の列オブジェクトの [行背景色配列](#) を使用して、両列から一つずつ、計二つの背景色を濃い青に変更します:

standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard

リストボックスを元の状態に戻すには、以下の手順でおこないます:

- 1、2 列目の行背景色配列の要素 2 に定数 `lk inherited` 定数を渡します。これにより行の赤の背景色を継承します。
- 5 列目の行スタイル配列の要素 3 と 4 に定数 `lk inherited` を渡します。これにより、要素 4 を除いて標準のスタイルを継承します (要素 4 はリストボックスの行スタイル配列にて指定されたイタリックの属性を継承します)。
- リストボックスの行スタイル配列の要素 4 に定数 `lk inherited` を渡します。これにより、4 行目のイタリックのスタイルが除去されます。
- リストボックスの行背景色配列の要素 2 に定数 `lk inherited` を渡します。これにより元の、背景色が交互に変わるリストボックスの状態に戻すことができます。

配列と式の使用

リストボックスのタイプに応じて、行のカラーやスタイル、表示について使用できるプロパティが異なります:

プロパティ	配列リストボックス	セレクションリストボックス	コレクションまたはエンティティセレクションリストボックス
背景色	行背景色配列	背景色式	背景色式" または メタ情報式
フォントカラー	行フォントカラー配列	フォントカラー式	フォントカラー式 または メタ情報式
フォントスタイル			

[行スタイル配列](#) | [スタイル式](#) | [スタイル式](#) または [メタ情報式](#) | 表示 | [行コントロール配列](#) | - | - |

リストボックスの印刷

リストボックスの印刷には 2つの印刷モードがあります: フォームオブジェクトのようにリストボックスを印刷する プレビュー モードと、フォーム内でリストボックスオブジェクトの印刷方法を制御できる 詳細 モード があります。フォームエディターで、リストボックスオブジェクトに "印刷" アピアランスを適用できる点に留意してください。

プレビューモード

プレビューモードでのリストボックスの印刷は、標準の印刷コマンドや 印刷 メニューを使用して、リストボックスを含むフォームをそのまま出力します。リストボックスはフォーム上に表示されている通りに印刷されます。このモードでは、オブジェクトの印刷を細かく制御することはできません。とくに、表示されている以上の行を印刷することはできません。

詳細モード

このモードでは、リストボックスの印刷は `Print object` コマンドを使用してプログラムにより実行されます (プロジェクトフォームとテーブルフォームがサポートされています)。 `LISTBOX GET PRINT INFORMATION` コマンドを使用してオブジェクトの印刷を制御できます。

このモードでは:

- オブジェクトの高さよりも印刷する行数が少ない場合、リストボックスオブジェクトの高さは自動で減少させられます ("空白" 行は印刷されません)。他方、オブジェクトの内容に基づき高さが自動で増大することはありません。実際に印刷されるオブジェクトのサイズは `LISTBOX GET PRINT INFORMATION` コマンドで取得できます。
- リストボックスオブジェクトは "そのまま" 印刷されます。言い換えれば、ヘッダーやグリッド線の表示、表示/非表示行など、現在の表示設定が考慮されます。これらの設定には印刷される最初の行も含みます。印刷を実行する前に `OBJECT SET SCROLL POSITION` を呼び出すと、リストボックスに印刷される最初の行はコマンドで指定した行になります。
- 自動メカニズムにより、表示可能な行以上の行数を含むリストボックスの印刷が容易になります。連続して `Print object` を呼び出し、呼び出し毎に別の行のまとめを印刷することができます。 `LISTBOX GET PRINT INFORMATION` コマンドを使用して、印刷がおこなわれている間の状態をチェックできます。

階層リストボックス

4D ではリストボックスを階層表示にするよう指定することができます。階層リストボックスは左の列が階層状に表示されます。このタイプの表示方法は、繰り返される値や、階層に依存するデータの表示などに適用できます (国/地域/都市など)。

[配列タイプ](#) のリストボックスのみを階層にできます。

階層リストボックスはデータを表示する特別な方法ですが、データの構造 (配列) は変更しません。階層リストボックスは通常のリストボックスとまったく同じ方法で管理されます。

階層の指定

階層リストボックスとして指定するには、3つの方法があります:

- フォームエディターのプロパティリストを使用して階層要素を手作業で設定する (または JSON フォームを編集する)。
- フォームエディターのリストボックス管理メニューを使用して階層を生成する。
- `LISTBOX SET HIERARCHY` や `LISTBOX GET HIERARCHY` コマンドを使用する (4D ランゲージリファレンス 参照)。

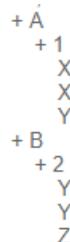
"階層リストボックス" プロパティによる階層化

このプロパティを使用してリストボックスの階層表示を設定します。JSON フォームにおいては、リストボックス列の *dataSource* プロパティの値が配列名のコレクションであるとき に階層化します。

階層リストボックス プロパティが選択されると、追加プロパティである Variable 1...10 が利用可能になります。これらには階層の各レベルとして使用するデータソース配列を指定します。これが *dataSource* の値である配列名のコレクションとなります。入力欄に値が入力されると、新しい入力欄が追加されます。10個までの変数を指定できます。これらの変数は先頭列に表示される階層のレベルを設定します。

Variable 1 は常に、リストボックスの先頭列の変数名に対応します (この 2つの値は自動でバインドされます)。Variable 1欄は常に表示され、入力できます。例: country。Variable 2 も常に表示され、入力できます。これは二番目の階層レベルを指定します。例: regions。三番目以降の欄は、その前の番号の欄が入力されると表示されます。例えば: counties、cities等。最大10レベルまで指定できます。ある階層レベルの値を削除すると、その後の階層レベルが繰り上がります。

最後の変数に複数の同じ値が存在しても、この変数が階層になることはありません。たとえば、arr1 に A A A B B B、arr2 に 1 1 1 2 2 2、そして arr3 に X X Y Y Y Z が値として設定されている場合、A、B、1、そして 2 は階層で表示できますが、X と Y は階層になりません:

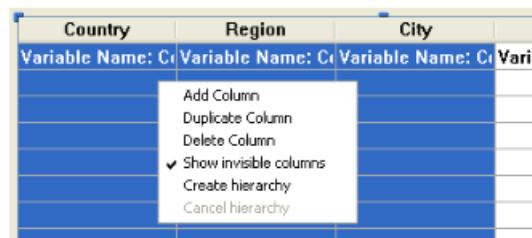


この原則は階層がひとつだけ設定されている場合には適用されません。この場合、同じ値はグループ化されます。

既存のリストボックスで階層を設定した場合、(最初のものを除き) これらの列を削除または非表示にしなければなりません。でないと、それらはリストボックス中で重複して表示されます。エディターのポップアップメニューを使用して階層を設定すると (階層リストボックス参照)、不要な列は自動でリストボックスから取り除かれます。

コンテキストメニューを使用した階層化

フォームエディター内で配列タイプのリストボックスオブジェクトの一番目から任意の数の列を選択すると、階層を作成 コマンドがコンテキストメニューから利用できるようになります:



このコマンドは階層化のショートカットです。このコマンドを選択すると、以下のアクションが実行されます:

- そのオブジェクトのプロパティリストで 階層リストボックス オプションが選択されます。
- その列の変数が階層を指定するために使用されます。既に設定されていた変数は置き換えられます。
- (先頭列を除き) 選択された列はリストボックス内に表示されなくなります。

例: 左から国、地域、都市、人口列が設定されたリストボックスがあります。国、地域、都市が (下図の通り) 選択され、コンテキストメニューから 階層を作成 を選択すると、先頭列に3レベルの階層が作成され、二番目と三番目の列は取り除かれます。人口列が二番目になります:

The screenshot shows a software interface with two main windows. On the left is a preview window titled 'PropertyList' containing a table with two columns: 'Country' and 'Population'. The 'Country' column has a variable name of 'Column1' and the 'Population' column has a variable name of 'Column4'. On the right is a configuration dialog for a 'List Box' object. The dialog includes sections for 'Objects' (Type: List Box, Object Name: List Box, Variable or Expression: List Box, Data Source: Arrays), 'List Box' (Number of Columns: 2, Number of Locked Columns: 0, Number of Static Columns: 1, Row Control Array: None, Selection Mode: Multiple), 'Headers' (Display Headers checked, Height: 1 lines), 'Footers' (Display Footers checked, Height: 1 lines), 'Gridlines' (unchecked), and 'Hierarchy' (Hierarchical List Box checked, Variable 1: Country, Variable 2: Region, Variable 3: City, Variable 4: None). Other sections like 'Coordinates & Sizing', 'Resizing Options', 'Entry', 'Display', 'Appearance', 'Background and Border', and 'Text' are also present.

階層をキャンセル

階層リストボックスとして定義されたリストボックスで先頭列を選択すると、階層をキャンセル コマンドを使用できます。このコマンドを選択すると以下のアクションが実行されます:

- そのオブジェクトの 階層リストボックス オプションの選択が解除されます。
 - 2番目以降の階層レベルが削除され、通常の列としてリストボックスに追加されます。

動作

階層リストボックスを含むフォームが最初に開かれる際、デフォルトですべての行が展開されています。

配列中で値が繰り返されていると、ブレーク行と階層 "ノード" がリストボックスに自動で追加されます。たとえば、リストボックスに都市に関する 4つの配列が含まれていて、それぞれ国、地域、都市名、人口データが含まれているとします:

リストボックスが階層形式で表示されると（先頭3つの配列が階層化されている場合）、以下のように表示されます：

City	Population
France	
Brittany	
Rennes	200000
Quimper	80000
Brest	120000
Normandy	
Caen	75000
Deauville	35000

階層を正しく構築するためには、事前に配列をソートしなければなりません。たとえば、配列中にデータが AAABBAACC の順で含まれていると、階層は以下のようになります:

```
> A B A C
```

階層 "ノード" を展開したり折りたたんだりするには、ノード上をクリックします。ノード上を Alt+クリック (Windows) または Option+クリック (macOS) すると、すべてのサブ要素が同時に展開されたり折りたたまれたりします。これらの動作は `LISTBOX EXPAND` および `LISTBOX COLLAPSE` コマンドを使用することでプログラミングでも実行可能です。

階層リストボックスに日付や時間型の値を表示する際、それらは `Short system format` で表示されます。

ソートの管理

階層モードのリストボックスにおいて、(リストボックス列のヘッダーをクリックして実行される) 標準の並べ替えは常に以下のようにおこなわれます:

- まず階層列 (一番目の列) のすべてのレベルが自動で昇順にソートされます。
- 次にクリックされた列の値を使用して、昇順または降順にソートが実行されます。
- すべての列が同期されます。
- その後の非階層列のソート時には、階層列の最後のレベルのみがソートされます。この列のソートはそのヘッダーをクリックすることでおこなえます。

例として、まだソートされていない以下のリストボックスがあります:

Country	Population
France	
Brittany	
Rennes	200000
Quimper	80000
Brest	120000
Lannion	20300
Lorient	35000
Normandy	
Caen	220000
Deauville	4000
Cherbourg	41000
Auvergne	
Vichy	27000
Moulins	20600
Belgium	
Wallonia	
Namur	111000
Liege	200000
Flanders	
Antwerp	472000
Louvain	95000
Brussels-Capital	
Brussels	155000

"Population" ヘッダーをクリックして人口に基づき昇順あるいは降順でソートを行おこなうと、データは以下のように表示されます:

1- Automatic sorting of all the hierarchical levels by ascending order

In the case of subsequent clicks on the Population header, only the last level will be synchronized

Country	Population	
Belgium		
Brussels-Capital		
Brussels	155000	
Flanders		
Antwerp	472000	
Louvain	495000	
Wallonia		
Liege	200000	
Namur	211000	
France		
Auvergne		
Vichy	27000	
Moulins	27600	
Brittany		
Rennes	200000	
Brest	220000	
Quimper	280000	
Lorient	285000	
Lannion	320300	
Normandy		
Caen	220000	
Cherbourg	441000	
Deauville	444000	

Click on the header: sorting of the population (ascending order)

2 - Sorting of the population by ascending order within the last level

通常のリストボックスと同様、リストボックスの **ソート可** オプションの選択を解除することで標準のソートメカニズムを無効にし、プログラムでソートを管理できます。

選択行とその位置の管理

階層リストボックスは、ノードの展開 / 折りたたみ状態により、スクリーン上に表示される行数が変わります。しかし配列の行数が変わるのはありません。表示が変わるだけでデータに変更はありません。この原則を理解することは重要です。階層リストボックスに対するプログラムによる管理は常に配列データに対しておこなわれるのであり、表示されたデータに対しておこなわれるわけではないからです。とくに、自動で追加されるブレーク行は、表示オプション配列では考慮されません（後述参照）。

例として以下の配列を見てみましょう：

France	Brittany	Brest
France	Brittany	Quimper
France	Brittany	Rennes

これらの配列が階層的に表示されると、2つのブレーク行が追加されるため、“Quimper” 行は 2行目ではなく 4行目に表示されます：

France

階層であっても、リストボックスにどのようにデータが表示されているかにかかわらず、“Quimper” が含まれる行を太字にしたい場合はステートメント `Style{2} = bold` を使用しなければなりません。配列中の行の位置のみが考慮されます。

この原則は以下のものを管理する内部的な配列に適用されます：

- カラー
- 背景色
- スタイル
- 非表示行
- 選択行

たとえば、Rennes を含む行を選択するには、以下のように書きます：

```
->MyListbox{3}:=True
```

非階層表示:

France	Brittany	Brest
France	Brittany	Quimper
France	Brittany	Rennes

階層表示:

France
Brittany
Brest
Quimper
Rennes

親が折りたたまれているため行が非表示になっていると、それらは選択から除外されます。(直接あるいはスクロールによって) 表示されている行のみを選択できます。言い換えれば、行を選択かつ隠された状態にすることはできません。

選択と同様に、LISTBOX GET CELL POSITION コマンドは階層リストボックスと非階層リストボックスにおいて同じ値を返します。つまり以下の両方の例題で、LISTBOX GET CELL POSITION は同じ位置 (3;2) を返します。

非階層表示:

France	Brittany	Brest	120000
France	Brittany	Quimper	80000
France	Brittany	Rennes	200000
France	Normandy	Caen	75000

階層表示:

France	
Brittany	
Brest	120000
Quimper	80000
Rennes	200000
Normandy	
Caen	75000

サブ階層のすべての行が隠されているとき、ブレーク行は自動で隠されます。先の例題で 1から 3行目までが隠されていると、"Brittany" のブレーク行は表示されません。

ブレーク行の管理

ユーザーがブレーク行を選択すると、LISTBOX GET CELL POSITION は対応する配列の最初のオカレンスを返します。以下のケースで:

France	
Brittany	
Brest	120000
Quimper	80000
Rennes	200000
Normandy	
Caen	75000

LISTBOX GET CELL POSITION は (2;4) を返します。プログラムでブレーク行を選択するには LISTBOX SELECT BREAK コマンドを使用する必要があります。

ブレーク行はリストボックスのグラフィカルな表示 (スタイルやカラー) を管理する内部的な配列では考慮されません。しかし、オブジェクトのグラフィックを管理するオブジェクト (フォーム) テーマのコマンドを使用してブレーク行の表示を変更できます。階層を構成する配列に対して、適切なコマンドを実行します。

以下のリストボックスを例題とします (割り当てた配列名は括弧内に記載しています):

非階層表示:

(T1)	(T2)	(T3)	(T4)	(tStyle)	(tColor)
France	Brittany	Brest	120000	Normal	0
France	Brittany	Quimper	80000	Underline	0
France	Brittany	Rennes	200000	Normal	0xFF0000
France	Normandy	Caen	220000	Normal	0
France	Normandy	Deauville	4000	Normal	0

階層表示:

France	
Brittany	
Brest	120000
Quimper	80000
Rennes	200000
Normandy	
Caen	75000
Deauville	4000

階層モードでは `tStyle` や `tColors` 配列で変更されたスタイルは、ブレーク行に適用されません。ブレークレベルでカラーやスタイルを変更するには、以下のステートメントを実行します:

```
OBJECT SET RGB COLORS(T1;0x0000FF;0xB0B0B0)
OBJECT SET FONT STYLE(T2;Bold)
```

このコンテキストでは、配列に割り当てられたオブジェクトがないため、オブジェクトプロパティコマンドで動作するのは、配列変数を使用したシンタックスのみです。

結果:

France	
Brittany	
Brest	120000
Quimper	80000
Rennes	200000
Normandy	
Caen	75000
Deauville	4000

展開/折りたたみ管理の最適化

`On Expand` や `On Collapse` フォームイベントを使用して階層リストボックスの表示を最適化できます。

階層リストボックスはその配列の内容から構築されます。そのためこれらの配列すべてがメモリにロードされる必要があります。大量のデータから (`SELECTION TO ARRAY` コマンドを使用して) 生成される配列をもとに階層リストボックスを構築するのは、表示速度だけでなくメモリ使用量の観点からも困難が伴います。

`On Expand` と `On Collapse` フォームイベントを使用することで、この制限を回避できます。たとえば、ユーザーのアクションに基づいて階層の一部だけを表示したり、必要に応じて配列をロード/アンロードできます。これらのイベントのコンテキストでは、`LISTBOX GET CELL POSITION` コマンドは、行を展開/折りたたむためにユーザーがクリックしたセルを返します。

この場合、開発者がコードを使用して配列を空にしたり値を埋めたりしなければなりません。実装する際注意すべき原則は以下のとおりです:

- リストボックスが表示される際、先頭の配列のみ値を埋めます。しかし 2番目の配列を空の値で生成し、リストボックスに展開/折りたたみアイコンが表示されるようにしなければなりません:

Artists/Albums/Tracks	CDs	Tracks	Durations
+ Brasil			
+ Celtic			
+ Classical			
+ Jazz			
+ New Age			
+ Others			
+ Pop/Rock			
+ Soundtrack			
+ World			

- ユーザーが展開アイコンをクリックすると `On Expand` イベントが生成されます。`LISTBOX GET CELL POSITION` コマンドはクリックされたセルを返すので、適切な階層を構築します: 先頭の配列に繰り返しの値を設定し、2番目の配列には `SELECTION TO ARRAY` コマンドから得られる値を設定します。そして `LISTBOX INSERT ROWS` コマンドを使用して必要なだけ行を挿入します。

Artists/Albums/Tracks	CDs	Tracks	Durations
+ Brasil			
+ Celtic			
+ Classical			
+ Jazz			
+ New Age			
+ Others			
+ Jacqueline Maillan			
+ Pierre Dac			
+ Pierre Dac & Francis Blanche			
+ Pop/Rock			
+ Soundtrack			
+ World			

- ユーザーが折りたたみアイコンをクリックすると `On Collapse` イベントが生成されます。 `LISTBOX GET CELL POSITION` コマンドはクリックされたセルを返すので、`LISTBOX DELETE ROWS` コマンドを使用してリストボックスから必要なだけ行を削除します。

オブジェクト配列の使用

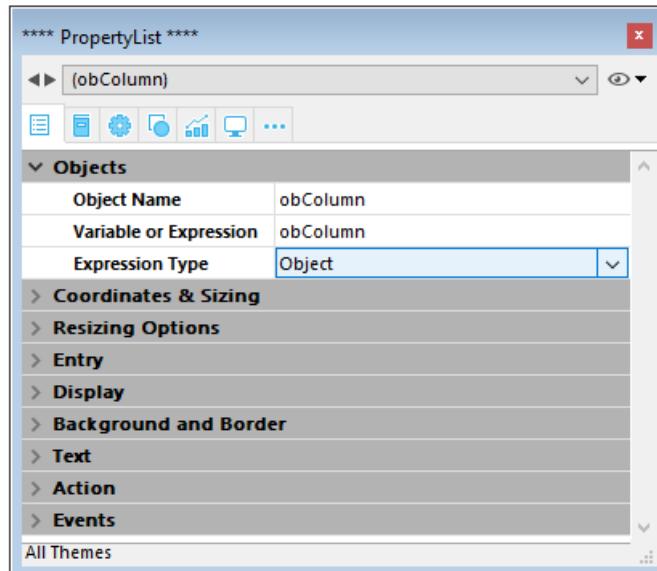
リストボックスのカラムはオブジェクト配列を扱えます。オブジェクト配列は異なる種類のデータを格納できるので、この強力な機能を使用すれば、单一のカラム内の行ごとに異なる入力タイプを混ぜたり、様々なウィジェットを表示したりといったことができるようになります。たとえば、最初の行にテキスト入力を挿入し、二行目にチェックボックスを、そして産業目次ドロップダウンを挿入する、と言ったことが可能になります。また、オブジェクト配列は、ボタンやカラーピッカーと言った新しいウィジェットへのアクセスも可能にします。

以下のリストボックスはオブジェクト配列を使用してデザインされました:

Label	Value
Document Name	MyReport
Document Type	PDF
Reference	123456
Category	
Include Abstract	<input checked="" type="checkbox"/>
Printable area size (height)	297 <input type="button" value="mm"/>
Printable area size (width)	210 <input type="button" value="mm"/>
Show Preview	<input type="button" value="Preview..."/>

オブジェクト配列カラムの設定

オブジェクト配列をリストボックスのカラムに割り当てるには、プロパティリスト（の "変数名" 欄）にオブジェクト配列名を設定するか、配列型のカラムのように `LISTBOX INSERT COLUMN` コマンドを使用します。プロパティリスト内では、カラムにおいて "式タイプ" にオブジェクトを選択できます:



オブジェクトカラムに対しては、座標、サイズ、スタイルなどに関連した標準のプロパティが使用可能です。プロパティリストを使用して定義する方法のほかにも、オブジェクト型のリストボックスカラムのそれぞれの行に対してスタイル、フォントカラー、背景色、表示状態をプログラムで定義することもできます。これらのタイプのカラムは非表示にすることも可能です。

しかしながら、データソーステーマは、オブジェクト型のリストボックスカラムに対しては選択できません。実際、カラムの各セルの中身は、それに対応するオブ

ジェクト配列の要素の属性に基づいています。配列の各オブジェクト要素には、以下を定義できます：

値の型 (必須): テキスト、カラー、イベント、他 値そのもの (任意): 入力/出力に使用 セルの内容表示 (任意): ボタン、リスト、他 追加の設定 (任意): 値の型によります これらのプロパティを定義するには、適切な属性をオブジェクト内に設定する必要があります (使用可能な属性は以下に一覧としてまとめてあります)。たとえば、以下のような簡単なコードを使用してオブジェクトカラム内に "Hello World!" 書き込むことができます：

```
ARRAY OBJECT(obColumn;0) // カラム配列
C_OBJECT($ob) // 第一要素
OB SET($ob;"valueType";"text") // 値の型を定義 (必須)
OB SET($ob;"value";"Hello World!") // 値を定義
APPEND TO ARRAY(obColumn;$ob)
```



表示フォーマットと入力フィルターはオブジェクトカラムに対しては設定できません。これらは値の型に応じて自動的に変わるからです。

valueTypeとデータ表示

リストボックスカラムにオブジェクト配列が割り当てられているとき、セルの表示・入力・編集の方法は、配列の要素の valueType 属性に基づきます。次の valueType の値がサポートされています：

- "text": テキスト値
- "real": セパレーターを含む数値。セパレーターの例: <space>, <.>, <,>
- "integer": 整数値
- "boolean": true/false 値
- "color": 背景色を定義
- "event": ラベル付ボタンを表示

4D は "valueType" の値に応じたデフォルトのウィジェットを使用します (つまり、"text" と設定すればテキスト入力ウィジェットが表示され、"boolean" と設定すればチェックボックスが表示されます)。しかし、オプションを使用することによって表示方法の選択が可能な場合もあります (たとえば、"real" と設定した場合、ドロップダウンメニューとしても表示できます)。以下の一覧はそれぞれの値の型に対してのデフォルトの表示方法と、他に選択可能な表示方の一覧を表しています：

valueType	デフォルトのウィジェット	他に選択可能なウィジェット
テキスト	テキスト入力	ドロップダウンメニュー (指定リスト) またはコンボボックス (選択リスト)
実数	管理されたテキスト入力 (数字とセパレーター)	ドロップダウンメニュー (指定リスト) またはコンボボックス (選択リスト)
integer	管理されたテキスト入力 (数字のみ)	ドロップダウンメニュー (指定リスト) またはコンボボックス (選択リスト) またはスリーステップチェックボックス
boolean	チェックボックス	ドロップダウンメニュー (指定リスト)
color	背景色	テキスト
event	ラベル付ボタン	
		すべてのウィジェットには、単位切り替えボタン または 省略ボタン を追加でセルに付属させることができます

セルの表示とオプションは、オブジェクト内の特定の属性を使用することによって設定できます (以下を参照ください)。

表示フォーマットと入力フィルター

オブジェクト型のリストボックスのカラムにおいては、表示フォーマットと入力フィルターを設定することはできません。これらは値の型に応じて自動的に定義されます。どのように定義されるかについては、以下一覧にまとめてあります：

値の型	デフォルトのフォーマット	入力コントロール
テキスト	オブジェクト内で定義されているものと同じ	制限なし
実数	オブジェクト内で定義されているものと同じ (システムの小数点セパレーターを使用)	"0-9" と "." と "-" min>=0 の場合、"0-9" と ":"
integer	オブジェクト内で定義されているものと同じ	"0-9" と "-" min>=0 の場合、"0-9" と ":"
ブール	チェックボックス	N/A
color	N/A	N/A
event	N/A	N/A

属性

オブジェクト配列の各要素は、セルの中身とデータ表示を定義する一つ以上の属性を格納するオブジェクトです（上記の例を参照ください）。

唯一必須の属性は "valueType" であり、サポートされる値は "text"、"real"、"integer"、"boolean"、"color" そして "event"です。以下の表には、リストボックスオブジェクト配列において "valueType" の値に応じてサポートされるすべての属性がまとめています（他の属性はすべて無視されます）。表示フォーマットに関しては、その更に下に詳細な説明と例があります。

	valueType	text	real	integer	boolean	color	event
属性	説明						
value	セルの値（入力または出力）	○	○	○			
min	最小値		○	○			
max	最大値		○	○			
behavior	"スリーステート" の値			○			
requiredList	オブジェクト内で定義されたドロップダウンリスト	○	○	○			
choiceList	オブジェクト内で定義されたコンボボックス	○	○	○			
requiredListReference	4D リスト参照 ("saveAs"の値による)	○	○	○			
requiredListName	4D リスト名 ("saveAs"の値による)	○	○	○			
saveAs	"reference" または "value"	○	○	○			
choiceListReference	4D リスト参照、コンボボックスを表示	○	○	○			
choiceListName	4D リスト名、コンボボックスを表示	○	○	○			
unitList	X要素の配列	○	○	○			
unitReference	選択された要素のインデックス	○	○	○			
unitsListReference	単位の4D リスト参照	○	○	○			
unitsListName	単位の4D リスト名	○	○	○			
alternateButton	切り替えボタンを追加	○	○	○	○	○	

value

セルの値は "value" 属性に保存されています。この属性は入力と出力に使用されるほか、リストを使用する際のデフォルト値を定義するのにも使用できます（以下参照）。

```

ARRAY OBJECT(obColumn;0) // カラム配列
C_OBJECT($ob1)
$entry:="Hello world!"
OB SET($ob1;"valueType";"text")
OB SET($ob1;"value";$entry) // ユーザーが新しい値を入力した場合、編集された値は $entry に格納されます
C_OBJECT($ob2)
OB SET($ob2;"valueType";"real")
OB SET($ob2;"value";2/3)
C_OBJECT($ob3)
OB SET($ob3;"valueType";"boolean")
OB SET($ob3;"value";True)

APPEND TO ARRAY(obColumn;$ob1)
APPEND TO ARRAY(obColumn;$ob2)
APPEND TO ARRAY(obColumn;$ob3)

```

Header1	
Hello	world!
0,66666666666667	
✓	

null 値はサポートされており、空のセルとして表示されます。

min と max

"valueType" が "real" または "integer" であるとき、min と max 属性もオブジェクトに設定できます（値は適切な範囲で、かつ、valueType と同じ型である必要があります）。

これらの属性を使用すると入力値の範囲を管理することができます。セルが評価されたとき（フォーカスを失ったとき）、入力された値が min の値より低い場合、または max の値より大きい場合には、その値は拒否されます。この場合、入力をする前の値が保持され、tip として説明が表示されます。

```

C_OBJECT($ob3)
$entry3:=2015
OB SET($ob3;"valueType";"integer")
OB SET($ob3;"value";$entry3)
OB SET($ob3;"min";2000)
OB SET($ob3;"max";3000)

```



behavior

behavior 属性は、値の通常の表示とは異なる表示方法を提供します。4D v15では、一つだけ他の表示方法が用意されています：

属性	使用可能な値	valueType	説明
behavior	threeStates	integer	スリーステートチェックボックスを数値として表現します。 2=セミチェック、1=チェック、0=チェックされていない、-1=非表示、-2=チェックなしが無効化、-3=チェックが無効化、-4=セミチェックが無効化

```

C_OBJECT($ob3)
OB SET($ob3;"valueType";"integer")
OB SET($ob3;"value";-3)
C_OBJECT($ob4)
OB SET($ob4;"valueType";"integer")
OB SET($ob4;"value";-3)
OB SET($ob4;"behavior";"threeStates")

```



requiredList と choiceList

"choiceList" または "requiredList" 属性がオブジェクト内に存在しているとき、テキスト入力は以下の属性に応じて、ドロップダウンリストまたはコンボボックスで置き換えられます：

- 属性が "choiceList" の場合、セルはコンボボックスとして表示されます。これはつまり、ユーザーは値を選択、または入力できるということです。
- 属性が "requiredList" の場合、セルはドロップダウンリストとして表示されます。これはつまり、ユーザーはリストに提供されている値からどれか一つを選択するしかないということです。

どちらの場合においても、"value" 属性を使用してウィジエット内の値を事前に選択することができます。

ウィジエットの値は配列を通して定義されます。既存の 4Dリストをウィジエットに割り当てたい場合、"requiredListReference"、"requiredListName"、"choiceListReference"、または "choiceListName" 属性を使用する必要があります。

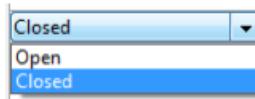
例：

- 選択肢が二つ ("Open" または "Closed") しかないドロップダウンリストを表示したい場合を考えます。デフォルトでは "Closed" が選択された状態にしたいとします：

```

ARRAY TEXT($RequiredList;0)
APPEND TO ARRAY($RequiredList;"Open")
APPEND TO ARRAY($RequiredList;"Closed")
C_OBJECT($ob)
OB SET($ob;"valueType";"text")
OB SET($ob;"value";"Closed")
OB SET ARRAY($ob;"requiredList";$RequiredList)

```

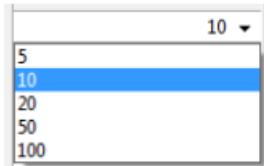


- 整数値であればすべて受け入れ可能な状態にしておいた上で、もっとも一般的な値を提示するためにコンボボックスを表示したい場合を考えます：

```

ARRAY LONGINT($ChoiceList;0)
APPEND TO ARRAY($ChoiceList;5)
APPEND TO ARRAY($ChoiceList;10)
APPEND TO ARRAY($ChoiceList;20)
APPEND TO ARRAY($ChoiceList;50)
APPEND TO ARRAY($ChoiceList;100)
C_OBJECT($ob)
OB SET($ob;"valueType";"integer")
OB SET($ob;"value";10) // 10 をデフォルト値として使用
OB SET ARRAY($ob;"choiceList";$ChoiceList)

```



requiredListName と requiredListReference

"requiredListName" と "requiredListReference" 属性を使用すると、デザインモード (ツールボックス内) またはプログラミングによって (`New list` コマンドを使用して) 4Dで定義されたリストをリストボックスセル内において使用することができるようになります。セルはドロップダウンリストとして表示されるようになります。これはつまり、ユーザーはリスト内に提供された値のどれか一つのみを選択できるということを意味します。

"requiredListName" または "requiredListReference" は、リストの作成元に応じて使い分けます。リストがツールボックスで作成された場合、リスト名を渡します。リストがプログラミングによって定義された場合は、リストの参照を渡します。どちらの場合においても、"value" 属性を使用してウィジェット内の値を事前に選択することができます。

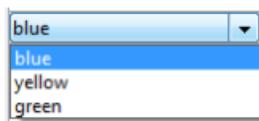
- これらの値を単純な配列を通して定義したい場合は、"requiredList" 属性を使用する必要があります。
- リストが実数値を表すテキストを含んでいる場合、小数点はローカル設定に関わらず、ピリオド (".") である必要があります。例: "17.6" "1234.456"

例:

- ツールボックスで定義された "colors" リスト ("blue"、"yellow"、そして "green" の値を格納) に基づいたドロップダウンリストを表示し、値として保存し、デフォルトの表示は "blue" にしたい場合を考えます:



```
C_OBJECT($ob)
OB SET($ob;"valueType";"text")
OB SET($ob;"saveAs";"value")
OB SET($ob;"value";"blue")
OB SET($ob;"requiredListName";"colors")
```



- プログラミングによって定義されたリストに基づいたドロップダウンリストを表示し、参照として保存したい場合を考えます:

```
<>List:=New list
APPEND TO LIST(<>List;"Paris";1)
APPEND TO LIST(<>List;"London";2)
APPEND TO LIST(<>List;"Berlin";3)
APPEND TO LIST(<>List;"Madrid";4)
C_OBJECT($ob)
OB SET($ob;"valueType";"integer")
OB SET($ob;"saveAs";"reference")
OB SET($ob;"value";2) // デフォルトでLondonを表示
OB SET($ob;"requiredListReference";<>List)
```

! [] (/docs/Rx/assets/en/FormObjects/listbox_column_objectArray_cities.png)

choiceListName と choiceListReference

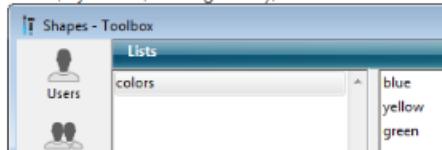
"choiceListName" と "choiceListReference" 属性を使用すると、デザインモード (ツールボックス内) またはプログラミングによって (New list コマンドを使用して) 4Dで定義されたリストをリストボックスセル内において使用することができるようになります。セルはコンボボックスとして表示されるようになります。これはつまり、ユーザーは値を選択、または入力できるということを意味します。

"choiceListName" または "choiceListReference" は、リストの作成元に応じて使い分けます。リストがツールボックスで作成された場合、リスト名を渡します。リストがプログラミングによって定義された場合は、リストの参照を渡します。どちらの場合においても、"value" 属性を使用してウィジェット内の値を事前に選択することができます。

- これらの値を単純な配列を通して定義したい場合は、"choiceList" 属性を使用する必要があります。
- リストが実数値を表すテキストを含んでいる場合、小数点はローカル設定に関わらず、ピリオド (".") である必要があります。例: "17.6" "1234.456"

例:

ツールボックスで定義された "colors" リスト ("blue"、"yellow"、そして "green" の値を格納) に基づいたドロップダウンリストを表示し、値として保存し、デフォルトの表示は "green" にしたい場合を考えます:



```
C_OBJECT($ob)
OB SET($ob;"valueType";"text")
OB SET($ob;"value";"blue")
OB SET($ob;"choiceListName";"colors")
```

unitsList、unitsListName、unitsListReference と unitReference

特定の値を使用することで、セルの値に関連した単位を追加することができます (例: "10 cm", "20 pixels" 等)。単位リストを定義するためには、以下の属性のどれか一つを使用します:

- "unitsList": 利用可能な単位 (例: "cm"、"inches"、"km"、"miles"、他) を定義するのに使用する x 要素を格納した配列。オブジェクト内で単位を定義するためには、この属性を使用します。
- "unitsListReference": 利用可能な単位を含んだ 4Dリストへの参照。New list コマンドで作成された 4D リストで単位を定義するためには、この属性を使用します。
- "unitsListName": 利用可能な単位を含んだデザインモードで作成された 4Dリスト名。ツールボックスで作成された 4Dリストで単位を定義するためには、この属性を使用します。

単位リストが定義された方法に関わらず、以下の属性を関連付けることができます:

- "unitReference": "unitList"、"unitsListReference" または "unitsListName" の値リスト内で選択された項目へのインデックス (1からx) を格納する单一の値。

カレントの単位は、ボタンとして表示されます。このボタンは、クリックするたびに "unitList"、"unitsListReference" または "unitsListName" の値を切り替えていきます (例: "pixels" -> "rows" -> "cm" -> "pixels" -> 等)。

例:

数値の入力と、その後に可能性のある二つの単位 ("lines" または "pixels") を続けて表示したい場合を考えます。カレントの値は "2" + "lines" と、オブジェクト内で直接定義された値 ("unitsList" 属性) を使用するものとします:

```
ARRAY TEXT($_units;0)
APPEND TO ARRAY($_units;"lines")
APPEND TO ARRAY($_units;"pixels")
C_OBJECT($ob)
OB SET($ob;"valueType";"integer")
OB SET($ob;"value";2) // 2 "units"
OB SET($ob;"unitReference";1) //"lines"
OB SET ARRAY($ob;"unitsList";$_units)
```

2 lines

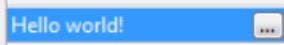
alternateButton

セルに省略ボタン [...] を追加したい場合、"alternateButton" 属性に true の値を入れてオブジェクトに渡すだけです。省略ボタンは自動的にセル内に表示されます。

このボタンがユーザーによってクリックされた場合、 On Alternate Click イベントが生成され、そのイベントを自由に管理することができます（詳細な情報に関しては [イベント管理](#) の章を参照ください）。

例：

```
C_OBJECT($ob1)
$entry:="Hello world!"
OB SET($ob;"valueType";"text")
OB SET($ob;"alternateButton";True)
OB SET($ob;"value";$entry)
```



color valueType

"color" valueType を使用すると、色、または色を表すテキストを表示することができます。

- 値が数字の場合、色付けされた長方形がセル内に表示されます。例：

```
C_OBJECT($ob4)
OB SET($ob4;"valueType";"color")
OB SET($ob4;"value";0x00FF0000)
```



- 値がテキストの場合、そのテキストが表示されます（例："value";"Automatic"）。

event valueType

"event" valueType を使用すると、クリックした際に On Clicked イベントを生成する単純なボタンを表示します。データまたは値を渡す/返すことはできません。

オプションとして、"label" 属性を渡すことができます。

例：

```
C_OBJECT($ob)
OB SET($ob;"valueType";"event")
OB SET($ob;"label";"Edit...")
```



イベント管理

オブジェクトリストボックス配列を使用している際には、複数のイベントを管理することができます：

- On Data Change: 以下の場所において、どんな値でも変更された場合には On Data Change イベントがトリガーされます：
 - テキスト入力
 - ドロップダウンリスト
 - コンボボックスエリア
 - 単位ボタン（値 x が値 x+1 へとスイッチしたとき）

- チェックボックス (チェック/チェックなしの状態がスイッチしたとき)
- On Clicked: ユーザーが、"event" valueType 属性を使用して実装されたボタンをクリックした場合、On Clicked イベントが生成されます。このイベントはプログラマーによって管理されます。
- On Alternative Click: ユーザーが省略ボタン ("alternateButton" 属性) をクリックした場合、On Alternative Click イベントが生成されます。このイベントはプログラマーによって管理されます。

ピクチャーボタン

ピクチャーボタンは [標準のボタン](#) と似ていますが、3つの状態（有効、無効、クリック）を持つ標準ボタンとは異なり、ピクチャーボタンでは、その名前が表すようにそれぞれの状態を別々のピクチャーにより表わします。

ピクチャーボタンは、次の2つの方法で使用します：

- フォーム上のコマンドボタンとして。この場合、ピクチャーボタンには通常4種類の状態があります（有効、無効、クリック、ロールオーバー）。たとえば、1行4列からなるサムネールテーブルの場合、各サムネールはデフォルト、クリック、ロールオーバー、無効という状態に対応しています。

プロパティ	JSON名	値
行	rowCount	1
列	columnCount	4
マウスアップで戻る	switchBackWhenReleased	true
ロールオーバー効果	switchWhenRollover	true
無効時に最終フレームを使用	useLastFrameAsDisabled	true

- 複数の選択項目の中からユーザーに選ばせるためのピクチャーボタンとして。この場合、ピクチャーボタンをポップアップピクチャーメニューの代わりに使用することができます。[ピクチャーポップアップメニュー](#) ではすべての選択肢が（ポップアップメニューの項目として）同時に表示されます。他方ピクチャーボタンは、選択候補を連続的に表示します（ボタンをクリックする度に変わります）。次に示すのは、ピクチャーボタンの例です。たとえば、カスタムアプリケーションのユーザーに、アプリケーションのインターフェース言語を選ばせたいものとします。そこで下図のように、選択候補をピクチャーボタンとしてカスタムプロパティダイアログボックスに組み込みます：



オブジェクトをクリックするとピクチャーが変わります。

ピクチャーボタンの使用

次の方法でピクチャーボタンを導入します。

- まず初めに1つの画像を用意し、一連のピクチャーを横、縦、または縦横の格子状に並べてその中に納めておきます。



ピクチャーは、縦、横、または縦横格子状に整理することができます（上図を参照）。ピクチャーボタンを格子状に並べた場合、各ピクチャーには上の行から順に左から右へと、0から始まる番号が振られます。たとえば、4行と3列で構成される格子において、2行目の2番目の画像の番号は4になります（上の例では英国旗）。

- プロジェクトのResourcesフォルダーに画像があるのを確認し、そのファイルパスを[パス名](#)プロパティに入力します。
- 画像の[行と列](#)プロパティを設定します。
- 画像の切り替え条件を[アニメーション](#)テーマのプロパティから選択します。

アニメーション

ピクチャーボタンは、標準的な配置や見た目の設定以外にも、表示モードと動作モードを指定する専用プロパティを設定することができます。これらのオプ

ションは組み合わせて使用することができます。

- デフォルト ([アニメーションオプション](#) 未選択) の場合、ユーザーがクリックすると、系列中の次のピクチャーを表示します。Shift キーを押しながらクリックすると、系列中の前のピクチャーを表示します。系列中の最後のピクチャーに到達すると、もう一度クリックしてもピクチャーは変わりません。つまりこの設定では、系列中の最初のピクチャーへ一巡して戻ることはできません。

次のモードを選択することができます：

- [先頭フレームに戻る](#)
- [マウスアップで戻る](#)
- [ロールオーバー効果](#)
- [マウス押下中は自動更新](#)
- [無効時に最終フレームを使用](#)
- [無効時に最終フレームを使用](#)

ピクチャーボタンに [関連付けた変数](#) は、ピクチャーのサムネールテーブルで現在表示されているピクチャーのインデックス番号を返します。このテーブル内のピクチャー番号は 0 から始まります。

プロパティ一覧

タイプ - オブジェクト名 - 変数あるいは式 - タイトル - CSSクラス - パス名 - 行 - 列 - マウス押下中は自動更新 - 先頭フレームに戻る - ロールオーバー効果 - マウスアップで戻る - 無効時に最終フレームを使用 - アニメーション間隔 (tick) - ボタンスタイル - 左 - 上 - 右 - 下 - 幅 - 高さ - 横方向サイズ変更 - 縦方向サイズ変更 - フォーカス可 - ショートカット - 表示状態 - 境界線スタイル - フォント - 太字 - イタリック - フォントカラー - ヘルプTips - 標準アクション - ドロップ有効

ピクチャー・ポップアップメニュー

ピクチャー・ポップアップメニューは、画像の二次元配列を表示するポップアップメニューです。ピクチャー・ポップアップメニューを使用して、[ピクチャー・ボタン](#) を置き換えることができます。ピクチャー・ポップアップメニューで使用するピクチャーの作成方法は、ピクチャーボタン用のピクチャーと似ています。その概念は [ボタングリッド](#) と同じですが、グラフィックがフォームオブジェクトではなくポップアップメニューとして使用される点が異なります。

ピクチャー・ポップアップメニューの使用

ピクチャー・ポップアップメニューを作成するには、[画像を参照](#) する必要があります。次の例は、ピクチャー・ポップアップメニューからインターフェース言語を選ぶことができます。各言語は対応する国旗で表わされています：



プログラミング

ピクチャー・ポップアップメニューは、メソッドを使用して管理できます。[ボタングリッド](#) と同様、ピクチャー・ポップアップメニューに割り当てられた変数に、選択された要素の値が代入されます。項目が選択されなければ、この値は 0 になります。各ピクチャーには上の行から順に左から右へと番号が振られます。

ページ指定アクション

ピクチャー・ポップアップメニューにページ指定 (`gotoPage`) [標準アクション](#) を割り当てることができます。このアクションを選択すると、4D はピクチャー配列で選択されたピクチャー位置に相当するフォームのページを自動的に表示します。要素は左から右、上から下に向かって番号が割り当てられます。

たとえば、ユーザーが3番目の要素をクリックすると、4D はカレントフォームの 3 ページ目 (存在する場合) を表示します。クリックをプログラムから管理したい場合は "動作なし" を選択します。

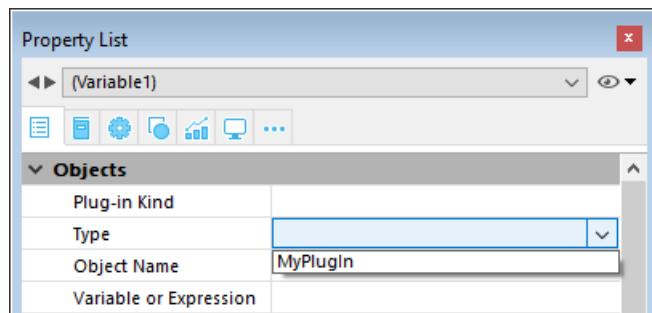
プロパティ一覧

[タイプ](#) - [オブジェクト名](#) - [変数あるいは式](#) - [CSSクラス](#) - [パス名](#) - [行](#) - [列](#) - [左](#) - [上](#) - [右](#) - [下](#) - [幅](#) - [高さ](#) - [横方向サイズ変更](#) - [縦方向サイズ変更](#) - [表示状態](#) - [境界線スタイル](#) - [太字](#) - [ヘルプTips](#) - [標準アクション](#)

プラグインエリア

プラグインエリアは、プラグインによりすべてが制御されるフォーム上のエリアです。フォームにプラグインエリアを追加できることで、カスタムアプリケーションを作成する際の可能性が限りなく広がります。プラグインでは、フォーム上にデジタル時計を表示するような単純な処理や、完全に機能するワープロ、スプレッドシート、グラフィック機能などを提供するなどの複雑な処理を実行できます。

アプリケーションを開く際、4Dは [アプリケーションにインストール](#) されたプラグインのリストを内部的に作成します。プラグインエリアをフォームに挿入すると、そのプロパティリスト内の タイプ リスト ("オブジェクト"テーマ内) にて、プラグインを割り当てることができます：



一部のプラグイン (4D Internet Commands など) はフォームやプラグインウインドウで使用することができません。プラグインがフォームで使用できない場合、そのプラグインはプロパティリストに候補として表示されません。

作成したプラグインエリアが小さすぎる場合、4Dはエリアをボタンとして表示し、エリアに割り当てられた変数名がそのタイトルに使用されます。実行時ユーザーはこのボタンをクリックしてプラグインを表示するためのウインドウを開くことができます。

詳細オプション

プラグインの作成者が詳細オプションを提供していると、プロパティリストのプラグイン テーマ内に [詳細設定](#) ボタンが使用可能になります。この場合ボタンをクリックすると、プラグインの制作元によるカスタムダイアログにてそれらのオプションを設定することができます。

プラグインのインストール

プラグインを 4D 環境にインストールするには、まず 4D を終了する必要があります。4D は起動時にプラグインをロードします。プラグインのインストールに関する詳細は [プラグインやコンポーネントのインストール](#) を参照してください。

プラグインの利用

独自にプラグインを作成したい場合、オープンソースのプラグイン制作キットを使用することができます。このキットの入手およびプラグイン作成に関する情報は ([github](#) 上にある) [完全なキット](#) を参照してください。

プロパティ一覧

タイプ - プラグインの種類 - オブジェクト名 - 変数あるいは式 - 式の型 - CSSクラス - 詳細設定 - 左 - 上 - 右 - 下 - 幅 - 高さ - 横方向サイズ変更 - 縦方向サイズ変更 - フォーカス可 - 表示状態 - 境界線スタイル - メソッド - ドラッグ有効 - ドロップ有効

進歩インジケーター

進歩インジケーター（または "サーモメーター"）は図形を用いて値を表示するオブジェクトです。



インジケーターの使用

インジケーターを使用して値の表示や設定ができます。たとえば、メソッドを用いてサーモメーターに値を指定すると、その値が表示されます。ユーザーがインジケーターポイントをドラッグすると、その値が変更されます。この値はフィールド、入力可オブジェクト、入力不可オブジェクト等の他のオブジェクトで使用することができます。

インジケーターに関連付けた変数により、その表示を管理します。メソッドを用いて、この変数に値を代入したり、またはインジケーターの値を使用したりします。たとえば、フィールドまたは入力可オブジェクトのメソッドを使用して、サーモメーターを管理できます：

```
vTherm:=[Employees]Salary
```

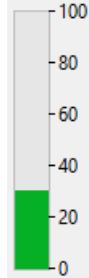
このメソッドは Salary フィールドの値を vTherm 変数に代入します。このメソッドは、たとえばフォーム上で Salary フィールドのオブジェクトメソッドとして記述できます。

逆にインジケーターを使用してフィールドの値を管理することも可能です。ユーザーはインジケーターをドラッグして値を設定します。このメソッドは次の通りです：

```
[Employees]Salary:=vTherm
```

このメソッドはインジケーターの値を Salary フィールドに代入します。ユーザーがインジケーターをドラッグすると、Salary フィールドの値が変わります。

デフォルトサーモメーター



サーモメーターはデフォルトの進歩インジケーターです。

縦または横のサーモメーターバーを表示できます。どちらになるかはフォームエディター上に描かれたオブジェクトの形により決定されます。

描画に関するプロパティをいくつか設定することができます：最小/最大値、目盛りの単位、ステップ、その他の表示オプションなどです。

プロパティ一覧

タイプ - オブジェクト名 - 変数あるいは式 ("整数", "数値", "日付", "時間" のみ) - 式の型 - CSSクラス - 最小 - 最大 - 目盛りのステップ - ステップ - ラベル位置 - 目盛りを表示 - バーバーショップ - 左 - 上 - 右 - 下 - 幅 - 高さ - 横方向サイズ変更 - 縦方向サイズ変更 - 入力可 - 数値フォーマット - 表示状態 - 境界線スタイル - フォント - フォントサイズ - 太字 - イタリック - 下線 - フォントカラー - ヘルプTips - オブジェクトメソッド実行

バーバーショップ

バーバーショップはデフォルトサーモメーターの一種です。このバリエーションを選択するには [バーバーショップ](#) プロパティをチェックします。

JSON コードにおいては、デフォルトサーモメーターのオブジェクトから "max" プロパティを取り除くだけで、インジケーターがバーバーショップになります。

バーバーショップは [スピナー](#) のように連続したアニメーションを表示します。このタイプのサーモメーターは通常プログラムが何らかの処理を行っていて、それが終了する時間が予測できない場合、そのことをユーザーに通知するために使用します。このバリエーションが選択されるとプロパティリストの [スケール](#) テーマは非表示になります。

フォームが実行されたとき、オブジェクトのアニメーションは開始されません。[割り当てられた変数](#) に値を代入してアニメーションを管理します：

- 非 0 値 = アニメーション開始
- 0 = アニメーション停止

プロパティ一覧

タイプ - オブジェクト名 - 変数あるいは式 ("整数", "数値", "日付", "時間" のみ) - 式の型 - CSSクラス - バーバーショップ - 左 - 上 - 右 - 下 - 幅 - 高さ - 横方向サイズ変更 - 縦方向サイズ変更 - 入力可 - 表示状態 - 境界線スタイル - フォント - フォントサイズ - 太字 - イタリック - 下線 - フォントカラー - ヘルプTips - オブジェクトメソッド実行

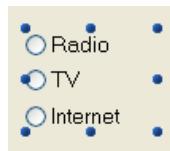
参照

- [ルーラー](#)
- [ステッパー](#)

ラジオボタン

ラジオボタンは、ボタングループの中から1つを選択することができるオブジェクトです。

ラジオボタンは通常、小さな目玉状の円とテキストを表示します。しかし、ラジオボタンに [それ以外の外観](#) を設定することもできます。



ラジオボタンを選択するには二つの方法があります:

- ラジオボタンをクリックする
- ラジオボタンにフォーカスがあるときに スペース キーを押す

ラジオボタンの設定

ラジオボタンは組織的な集合の中で使用されます。その集合のなかから一度にひとつのボタンしか選択できません。一連のラジオボタンが連携した状態で動作するためには、それらには同じ [ラジオグループ](#) プロパティが設定されていなくてはなりません。

ラジオボタンの結果はメソッドを用いて管理します。あらゆるボタンと同様に、フォームが初めて開かれる時ラジオボタンは 0 に初期化されています。ラジオボタンが選択されると、ラジオボタンに割り当てられたメソッドが実行されます。次の例では、ビデオ収集データベースでラジオボタンを使用し、レコーディングの速さ (SP, LP, EP) を入力します:



グループのなかから 1 つのラジオボタンを選択すると、そのボタンには 1 が代入され、グループ内の他のすべてのボタンには 0 が代入されます。一度に1つのラジオボタンしか選択できません。

ラジオボタンには [ブール型の式](#) を設定することができます。この場合、グループ内で選択されたラジオボタンの変数には true が代入され、残りのラジオボタンの変数には false が代入されます。

ラジオボタンオブジェクトに格納された値は (ブールフィールドの場合を除き) 自動保存されません。変数に格納されたラジオボタンの値はメソッドで管理しなければなりません。

ボタンスタイル

ラジオ [ボタンスタイル](#) は、ラジオボタンの外観を制御すると同時に、提供されるプロパティをも決定します。ラジオボタンには、あらかじめ定義されたスタイルを割り当てることができます。しかし、ラジオボタンが適切に動作するには、同じグループに所属するラジオボタンはすべて同じボタンスタイルに設定されている必要があります。

次の既定スタイルが提供されています:

通常

通常スタイルのラジオボタンは、標準的なシステムボタンで (小さな目玉状の円とテキストを表示したもの)、ユーザークリックに応じてコードを実行します。



通常スタイルのラジオボタンにマウスオーバーすると、"目玉" の色が変化します。

フラット

フラットスタイルのラジオボタンは、標準的なシステムボタンで（小さな目玉状の円とテキストを表示したもの）、ユーザークリックに応じてコードを実行します。



フラットスタイルでは、装飾が最小限に抑えられています。フラットボタンのグラフィック的な装飾は最小限であるため、印刷されるフォームでの使用に適しています。

ツールバー

ツールバースタイルのラジオボタンは、主としてツールバーで使用するためのものです。

ツールバー ボタンは、透明の背景に中央配置のラベルがデフォルトで付いています。ボタンにマウスオーバーしたときの表示は OS によって異なります：

- Windows - ボタンがハイライト表示されます。



- macOS - ボタンはハイライト表示されません。

ベベル

ベベルスタイルは [ツールバー](#) スタイルと似た動作をしますが、薄いグレーの背景にグレーの枠が描画されます。ボタンにマウスオーバーしたときの表示は OS によって異なります：

- Windows - ボタンがハイライト表示されます。



- macOS - ボタンはハイライト表示されません。

角の丸いベベル

角の丸いベベルスタイルは [ベベル](#) スタイルとほぼ同一ですが、OSによっては角が丸く表示されます。

- Windows 上では、このスタイルは [ベベル](#) スタイルと同じです。
- macOS - 角が丸くなっています。



OS Xグラデーション

OS Xグラデーションスタイルは [ベベル](#) スタイルとほぼ同一ですが、OSによっては異なる点があります。

- Windows 上では、このスタイルは [ベベル](#) スタイルと同じです。
- macOS - 2トーンのシステムボタンです。

OS Xテクスチャー

OS Xテクスチャースタイルは [ツールバー](#) スタイルとほぼ同一ですが、OSによってはマウスオーバー時の変化がないほか、外観の異なる点があります。

デフォルトで、OS Xテクスチャーボタンの外観は次の通りです：

- Windows - ツールバースタイルのようなボタンに中央配置のラベルが付き、背景は常に表示されます。
- macOS - 灰色のグラデーションを表示する標準のシステムボタンです。高さは定義済みで、変更できません。

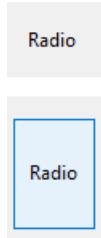


Office XP

Office XPスタイルは通常ボタン（標準のシステムボタン）のような外観に、[ツールバー](#)ボタンスタイルの動作を組み合わせたものです。

Office XPボタンの反転表示と背景のカラーはシステムカラーに基づいています。ボタンにマウスオーバーしたときの表示は OS によって異なります：

- Windows - マウスオーバー時にのみ背景が表示されます。



- macOS - 背景は常に表示されます。

折りたたみ/展開

このスタイルは標準の折りたたみ/展開アイコンを表示するのに使用します。これらは階層リストで使用されます。Windows では [+] または [-] のように表示されます。macOS では、右や下を指す三角として表示されます。



開示ボタン

開示ボタンスタイルが適用されると、詳細情報の表示/非表示にするのに使われる標準的な開示ボタンとして描画されます。値が 0 のときはボタンの矢印が下向き、値が 1 のときは上向きになります。



カスタム

カスタムスタイルのラジオボタンは、背景ピクチャーを使用できるほか、さまざまな追加パラメーターを管理することができます ([アイコンオフセット](#) や [マージン](#))。

プロパティ一覧

すべてのラジオボタンは次の基本プロパティを共有します：

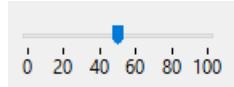
タイプ - オブジェクト名 - 変数あるいは式 - 式の型 - タイトル - ラジオグループ - 値を記憶 - CSSクラス - ボタンスタイル - 左 - 上 - 右 - 下 - 幅 - 高さ - 横方向サイズ変更 - 縦方向サイズ変更 - フォーカス可 - ショートカット - 表示状態 - フォント - 太字 - イタリック - 下線 - フォントカラー - ヘルプTips - メソッド

[ボタンスタイル](#) に応じて、次の追加プロパティが使用できます：

- [背景パス名](#) - [アイコンオフセット](#) - [横方向マージン](#) - [縦方向マージン](#) (カスタムスタイル)
- [ピクチャーパス名](#) - [状態の数](#) - [タイトル/ピクチャー位置](#) (ツールバー・ボタン、ベベル、角の丸いベベル、OS X グラデーション、OS X テクスチャー、Office XP、カスタム)

ルーラー

ルーラーは、カーソルを使用して目盛り上を移動することで値を選択する標準のインターフェースオブジェクトです。



割り当てられた変数 の値を入力エリア (フィールドまたは変数) に代入して格納したり、逆にオブジェクトの現在値を設定することもできます。

詳細については [インジケーターの使用](#) を参照ください。

プロパティ一覧

タイプ - オブジェクト名 - 変数あるいは式 - 式の型 - CSSクラス - 最小 - 最大 - 目盛りのステップ - ステップ - ラベル位置 - 目盛りを表示 - 左 - 上 - 右 - 下 - 幅 - 高さ - 横方向サイズ変更 - 縦方向サイズ変更 - 入力可 - 数値フォーマット - 表示状態 - 境界線スタイル - 太字 - ヘルプ
Tips - オブジェクトメソッド実行

参照

- [進捗インジケーター](#)
- [ステッパー](#)

図形

図形は、4D フォームに設置することができる [スタティックオブジェクト](#) です。

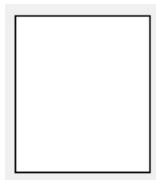
次の基本の図形が提供されています:

- 四角
- 線
- 楕円

四角

フォーム上に四角を配置することで、視覚的な効果が得られます。四角で描画できるのは長方形に限られます。

四角のグラフィック属性（線カラー、線幅、点線タイプ等）やリサイズオプションはプロパティリストにて指定できます。角の [丸み](#) を指定することもできます。



JSON 例:

```
"myRectangle": {  
    "type": "rectangle",      // オブジェクトタイプ  
    "left": 60,                // フォーム上の座標（左）  
    "top": 160,                // フォーム上の座標（上）  
    "width": 100,               // 幅  
    "height": 20,                // 高さ  
    "borderRadius": 20        // 角の半径（丸み）  
}
```

プロパティ一覧

タイプ - オブジェクト名 - CSSクラス - 左 - 上 - 右 - 下 - 幅 - 高さ - 角の半径 - 横方向サイズ変更 - 縦方向サイズ変更 - 表示状態 - 塗り
カラー - 線カラー - 線幅 - 点線タイプ

線

フォーム上に線を配置することで、視覚的な効果が得られます。線は水平、垂直のほか、あらゆる角度で描画することができます。

線のグラフィック属性（線カラー、線幅、点線タイプ等）やリサイズオプションはプロパティリストにて指定できます。

startPoint プロパティ

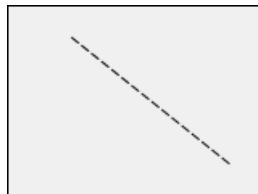
`startPoint` JSON プロパティは、線の始点を定義します (JSON例参照)。

フォームエディター上では線の始点があきらかなため、プロパティリストにおいて `startPoint` プロパティは非表示です。

JSON 例:

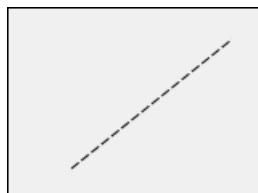
```
"myLine": {  
    "type": "line",  
    "left": 20,  
    "top": 40,  
    "width": 100,  
    "height": 80,  
    "startPoint": "topLeft", // 第一の方向  
    "strokeDashArray": "6 2" // 破線  
}
```

結果:



```
"myLine": {  
    "type": "line",  
    "left": 20,  
    "top": 40,  
    "width": 100,  
    "height": 80,  
    "startPoint": "bottomLeft", // 第二の方向  
    "strokeDashArray": "6 2" // 破線  
}
```

結果:

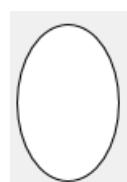


プロパティ一覧

タイプ - オブジェクト名 - CSSクラス - 左 - 上 - 右 - 下 - 幅 - 高さ - 横方向サイズ変更 - 縦方向サイズ変更 - 表示状態 - 線カラー - 線幅
- 点線タイプ - startPoint

楕円

フォーム上に楕円を配置することで、視覚的な効果が得られます。楕円を使って円を描くことができます（幅と高さを同じ値に設定します）。



JSON 例:

```
"myOval": {  
    "type": "oval",           // オブジェクトタイプ  
    "left": 60,              // フォーム上の座標（左）  
    "top": 160,              // フォーム上の座標（上）  
    "width": 100,             // 幅  
    "height": 20,             // 高さ  
    "fill": "blue"            // 塗りカラー  
}
```

プロパティ一覧

タイプ - オブジェクト名 - CSSクラス - 左 - 上 - 右 - 下 - 幅 - 高さ - 横方向サイズ変更 - 縦方向サイズ変更 - 表示状態 - 塗りカラー - 線カラー - 線幅 - 点線タイプ

スピナー

スピナーは円形のインジケーターで、[バーバーショップ](#) のように連続したアニメーションを表示します。



このタイプのオブジェクトは通常、ネットワークへの接続や、計算などの処理が実行中であることを表すために使用します。このインジケーターが選択されるとプロパティリストの [スケール](#) テーマは非表示になります。

フォームが実行されたとき、オブジェクトのアニメーションは開始されません。[割り当てられた変数](#) に値を代入してアニメーションを管理します：

- 非 0 値 = アニメーション開始
- 0 = アニメーション停止

プロパティ一覧

[タイプ](#) - [オブジェクト名](#) - [変数あるいは式](#) - [式の型](#) - [CSSクラス](#) - [左](#) - [上](#) - [右](#) - [下](#) - [幅](#) - [高さ](#) - [横方向サイズ変更](#) - [縦方向サイズ変更](#) - [表示状態](#) - [境界線スタイル](#) - [ヘルプTips](#)

スプリッター

スプリッターはフォームを2つのエリアに分割します。ユーザーはいずれかの方向へスプリッターを移動してエリアを拡げたり縮めたりすることができます。水平方向または垂直方向のスプリッターを作成できます。スプリッターでは各オブジェクトのサイズ調整プロパティが考慮されます。つまり作成するアプリケーションのインターフェースをすべてカスタマイズすることができます。また、スプリッターは“プッシュ”（押し込みタイプ）にすることも可能です。

たとえば、スプリッターは列のサイズを変更できるよう、リストフォームで使用されます：

Job Title:	Company:
Secretary	Howard Battery Co.
Salesperson	Howard Battery Co.
Salesperson	Howard Battery Co.
Supervisor	Howard Battery Co.
Director	BluePines

スプリッターの一般的な特徴をいくつか次に説明します：

- あらゆるタイプのフォーム上にスプリッターを必要なだけ設置可能であり、一つのフォーム上で水平と垂直のスプリッターと一緒に使用することができます。
- スプリッターはオブジェクトを横切ることができます（オーバーラップ）。スプリッターを動かすと、このオブジェクトのサイズが変更されます。
- フォーム上で移動されたオブジェクトが完全に表示されたままになるように、また別のスプリッターを超えないように、スプリッターの停止位置が計算されます。[以降のオブジェクトを移動する](#)（プッシュ）プロパティをスプリッターに割り当てると、スプリッターを右方向または下方向へ動かしても停止することはできません。
- スプリッターを使用するフォームのサイズを変更すると、フォームが表示されている間だけ、フォームの新しいサイズが保存されます。フォームを閉じると、最初の大きさに戻ります。

スプリッターは挿入されると線として表示されます。その [線のスタイル](#) を変更してさらに細い線に設定したり、線の種類によっては [線の色](#) を設定したりすることができます。

JSON 例：

```
"mySplitter": {  
    "type": "splitter",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20,  
    "splitterMode": "move" // プッシュ  
}
```

プロパティ一覧

[タイプ](#) - [オブジェクト名](#) - [変数あるいは式](#) - [CSSクラス](#) - [左](#) - [上](#) - [右](#) - [下](#) - [幅](#) - [高さ](#) - [横方向サイズ変更](#) - [縦方向サイズ変更](#) - [以降のオブジェクトを移動する](#) - [表示状態](#) - [境界線スタイル](#) - [線カラー](#) - [ヘルプTips](#)

隣接するオブジェクトのプロパティとの相互作用

フォーム上では、スプリッター周辺にある各オブジェクトのリサイズオプションに基づいて、スプリッターとこれらのオブジェクトとが作用しあいます：

オブジェクトのリサイズオプション	水平スプリッターの上、または垂直スプリッターの左にあるオブジェクト(1)	水平スプリッターの下、または垂直スプリッターの右にあるオブジェクト (非 "プッシャー" の場合)	水平スプリッターの下、または垂直スプリッターの右にあるオブジェクト ("プッシャー" の場合)
なし	そのまま変わらない	次に停止するまでスプリッターとともに移動 (スプリッターとの相対的な位置は変更されない)。下または右への移動時の停止位置はウインドウの境界または別のスプリッター。	無制限にスプリッターとともに移動 (スプリッターとの相対的な位置は変更されない)。停止しない (次節を参照)。
拡大	元の位置のままだが、スプリッターの新しい位置に基づいてサイズが調整される		
移動	スプリッターとともに移動する		

(1) この位置にあるオブジェクトを超えて、右側 (水平)、または下側 (垂直) ヘスプリッターをドラッグすることはできません。

スプリッターが定義される矩形内にすべて納まるオブジェクトは、スプリッターと一緒に移動します。

プログラムによるスプリッターの管理

オブジェクトメソッドをスプリッターに指定することができます。スプリッターを移動する間 `On Clicked` イベントでこのメソッドが呼び出されます。

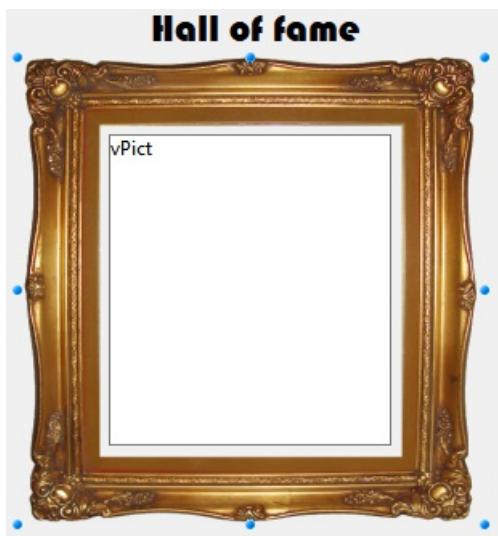
各スプリッターには 倍長整数 型の **変数** が割り当てられます。この変数はオブジェクトメソッドやフォームメソッドで使用することができます。この変数の値により、スプリッターの最初の位置に対する現在位置がピクセル単位で示されます。

- この値が負数の場合: スプリッターは上または左方向へ移動されました。
- この値が正数の場合: スプリッターは下または右方向へ移動されました。
- この値が 0 の場合: スプリッターは元の位置に移動されました。

プログラムによってスプリッターを移動させることも可能です。これをおこなうには、関連付けた変数の値を設定します。たとえば、垂直スプリッターに `split1` という名前の変数が関連付けられている場合、`split1:=-10` という命令を実行すると、ユーザーが手動で動かす場合と同じように、スプリッターは 10ピクセル左方向へ移動します。移動が実際におこなわれるのは、この命令文が記述されているフォームメソッドまたはオブジェクトメソッドを実行しあわったときです。

スタティックピクチャー

スタティックピクチャーは、4D フォームに設置することができる [スタティックオブジェクト](#) で、フォームの装飾や、背景、ユーザーインターフェースなどの目的で使用されます。



スタティックピクチャーはフォーム外に格納され、参照によって挿入されます。フォームエディターにおいては、コピーペーストやドラッグ & ドロップ操作によってスタティックピクチャーオブジェクトが作成されます。

複数ページあるフォームの 0 ページにピクチャーを配置すると、そのピクチャーは自動ですべてのページに背景として表示されます。またピクチャーを継承フォームに置くこともでき、そのフォームを継承するすべてのフォームの背景とすることもできます。どちらの方法も、それぞれのページにピクチャーを配置するより、アプリケーションの動作が速くなります。

形式と保存場所

ピクチャーは 4D がネイティブに管理するフォーマットでなければなりません (4D は主なピクチャーフォーマットを認識します: JPEG, PNG, BMP, SVG, GIF, 等)。

ピクチャーパスに指定できる場所は次の 2箇所です:

- プロジェクトの Resources フォルダー。プロジェクト内の複数のフォームで画像を共有する場合に適切です。この場合、パス名は "/RESOURCES/<picture path>" となります。
- フォームフォルダー内の画像用フォルダー (たとえば、Images と名付けたフォルダー)。特定のフォームでしか画像が使われない場合や、そのフォームの全体を複製してプロジェクト内、または別のプロジェクトに移動させたい場合に適切です。この場合、パス名は "<¥picture path>" となり、フォームフォルダーを基準とした相対パスです。

プロパティー一覧

タイプ - オブジェクト名 - CSSクラス - パス名 - 表示 - 左 - 上 - 右 - 下 - 幅 - 高さ - 横方向サイズ変更 - 縦方向サイズ変更 - 表示状態

ステッパー

このオブジェクトを使用すると、ユーザーは矢印ボタンをクリックすることで、数値、時間、または日付を定義済みのステップ毎にスクロールすることができます。

Stepper associated with `vStep` variable



ステッパーの使用

[割り当てられた変数](#) の値を入力エリア (フィールドまたは変数) に代入して格納したり、逆にオブジェクトの現在値を設定することもできます。

ステッパーには数値、時間、日付変数を割り当てることができます。

- 時間型の値では最小、最大、[ステップ](#) プロパティは秒を表します。たとえば、ステッパーを 8:00 から 18:00 まで 10 分のステップで設定するには:
 - 最小** = 28,800 ($8 * 60 * 60$)
 - 最大** = 64,800 ($18 * 60 * 60$)
 - ステップ** = 600 ($10 * 60$)
- 日付タイプの値では [ステップ](#) プロパティに入力された値が日数を表します。最小と最大プロパティは無視されます。

ステッパーを時間や日付変数に対して動作させるためには、プロパティリストで型を設定するだけでなく、[C_TIME](#) または [C_DATE](#) コマンドで明示的に宣言する必要があります。

詳細については [インジケーターの使用](#) を参照ください。

プロパティ一覧

[タイプ](#) - [オブジェクト名](#) - [変数あるいは式](#) - [式の型](#) - [CSSクラス](#) - [最小](#) - [最大](#) - [ステップ](#) - [左](#) - [上](#) - [右](#) - [下](#) - [幅](#) - [高さ](#) - [横方向サイズ変更](#)
- [縦方向サイズ変更](#) - [入力可](#) - [表示状態](#) - [境界線スタイル](#) - [ヘルプTips](#) - [オブジェクトメソッド実行](#)

参照

- [進捗インジケーター](#)
- [ルーラー](#)

サブフォーム

サブフォームとは、他のフォームに組み込まれるフォームのことです。

用語

サブフォームに実装されたコンセプトを明確に説明するために、いくつかの用語についてここで定義します：

- サブフォーム：他のフォームに組み込まれることを意図したフォーム。
- 親フォーム：1つ以上のサブフォームを含むフォーム。
- サブフォームコンテナー：親フォームに組み込まれた、サブフォームのインスタンスを表示するオブジェクト。
- サブフォームインスタンス：親フォームに表示されたサブフォームの実体。このコンセプトはとても重要です。親フォームには、同じサブフォームのインスタンスを複数表示することができるからです。
- リストフォーム：データをリストとして表示するサブフォームインスタンス。
- 詳細フォーム：リストサブフォームをダブルクリックすることでアクセスすることができる、ページタイプの入力フォーム。

リストサブフォーム

リストサブフォームを使うことで、他のテーブルのデータを入力、表示、および更新することができます。通常は 1対Nリレーションが設定されたデータベースでリストサブフォームを使用します。リレートされた 1テーブルのフォーム上のリストサブフォームでは、リレートされた Nテーブルのデータを入力・表示・更新します。一つのフォーム上に、それぞれ異なるテーブルに属する複数のサブフォームを配置できます。しかし、フォームの同じページ上には、同じテーブルに属するサブフォームを複数配置することはできません。

たとえば、連絡先管理データベースでは、ある連絡先のすべての電話番号を表示するためにリストサブフォームが使用されるでしょう。連絡先テーブルの画面に電話番号が表示されますが、情報はリレートテーブルに格納されています。1対Nリレーションを使用することで、このデータベース設計は連絡先ごとに複数の電話番号を簡単に格納できるようになっています。自動リレーションにより、リレートされている Nテーブルへのデータ入力がプログラムなしで直接おこなうことができます。

リストサブフォームは通常 Nテーブルに結び付けられますが、それだけでなく他の任意のデータベーステーブルのコードをサブフォームのインスタンスに表示することもできます。

また、ユーザーがリストサブフォームに直接データを入力するようにもできます。サブフォームの設定に基づき、ユーザーがサブレコード上でダブルクリックするか、サブレコードを追加/編集するコマンドを使用すると、詳細フォームが表示されます。

4Dはサブレコードを管理する基本的なニーズに応える 3つの標準アクション `editSubrecord` (サブレコード編集)、`deleteSubrecord` (サブレコード削除) および `addSubrecord` (サブレコード追加) を提供しています。フォームに複数のサブフォームインスタンスが含まれる場合、フォーカスを持っているサブフォームにアクションが適用されます。

ページサブフォーム

ページサブフォームは、カレントサブレコードのデータや、コンテキストに基づく関連する値（変数やピクチャーなど）を表示できます。ページサブフォームを使用する利点の一つは、それらが高度な機能を提供したり、親フォームと相互作用したりできることです（ウィジェット）。ページサブフォームには専用のプロパティやイベントがあり、プログラムから完全に制御することができます。

ページサブフォームは [詳細フォーム](#) プロパティで指定された入力フォームを使用します。リストサブフォームと異なり、使用されるフォームは親フォームと同じテーブルに所属していてもかまいません。また、プロジェクトフォームを使用することもできます。実行時、ページサブフォームは入力フォームと同じ標準の表示特性を持ちます。

4D ウィジェットは、ページサブフォームに基づいた定義済みの複合オブジェクトです。詳細は専用のドキュメント [4D Widgets \(ウィジェット\)](#) を参照してください。

バインドされた変数あるいは式の管理

サブフォームコンテナーオブジェクトには、[変数あるいは式](#) をバインドすることができます。これは、親フォームとサブフォーム間で値を同期するのに便利で

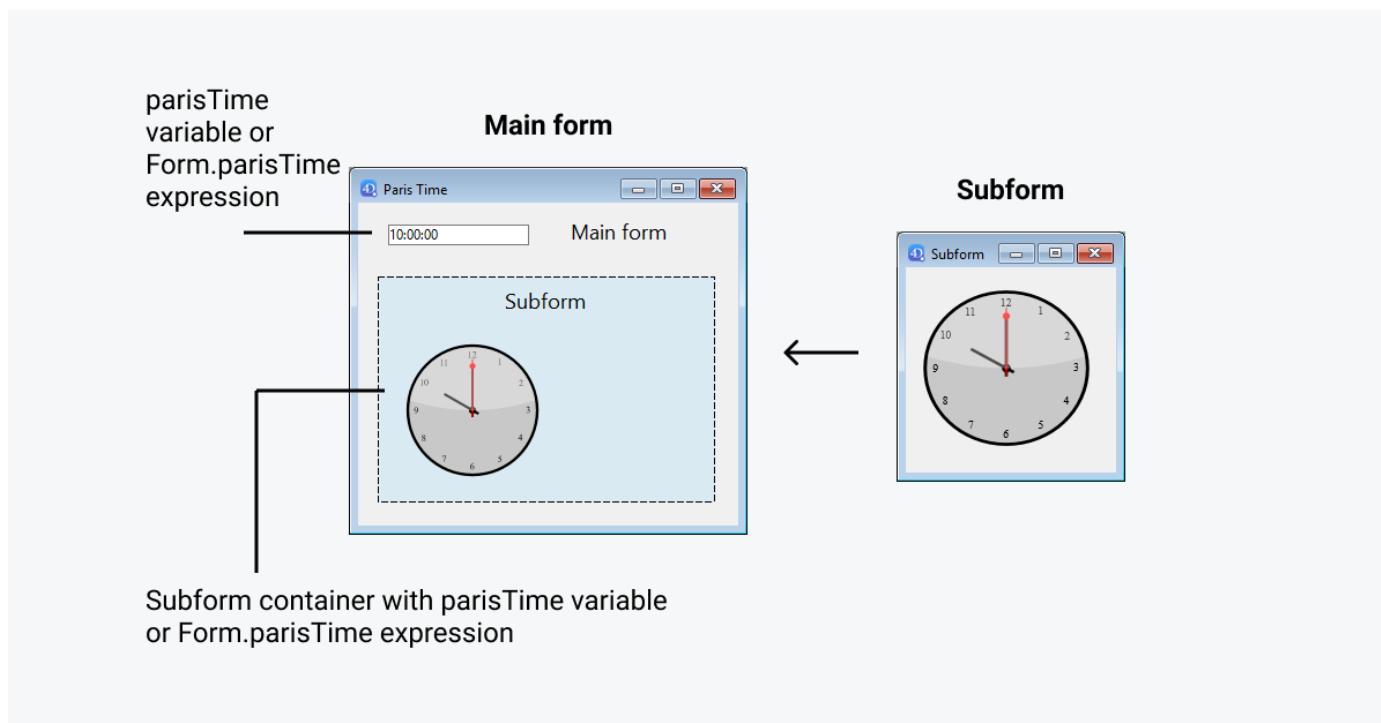
す。

デフォルトで、4D はサブフォームコンテナーに オブジェクト型 の変数あるいは式をバインドし、Form コマンドを使ってサブフォームのコンテキストで値を共有できるようにします。しかし、単一の値のみを共有したい場合は、任意のスカラー型（時間、整数など）の変数や式を使用することもできます。

- バインドするスカラー型の変数あるいは式を定義し、On Bound Variable Change や On Data Change フォームイベントが発生したときに、OBJECT Get subform container value や OBJECT SET SUBFORM CONTAINER VALUE コマンドを呼び出して値を共有します。この方法は、単一の値を同期させるのに推奨されます。
- または、バインドされた オブジェクト 型の変数あるいは式を定義し、Form コマンドを使用してサブフォームからそのプロパティにアクセスします。この方法は、複数の値を同期させるのに推奨されます。

親フォームとサブフォームの同期（単一値）

親フォームにおいて、他のオブジェクトとサブフォームコンテナーと同じ "変数あるいは式" を設定することで、親フォームとサブフォームのコンテキストをリンクし、洗練されたインターフェースを作成することができます。たとえば、入力可の時間型変数が置かれている親フォームに、アナログ時計を表示するサブフォームを置くとします：



親フォームにおいて、両オブジェクト（時間変数とサブフォームコンテナー）の 変数あるいは式 プロパティは同じ設定にされています。変数（例： parisTime ）、あるいは式（例： Form.parisTime ）を設定することができます。

サブフォームにおいては、時計オブジェクトの同プロパティとして Form.clockValue が設定されています。

サブフォームの内容を更新する

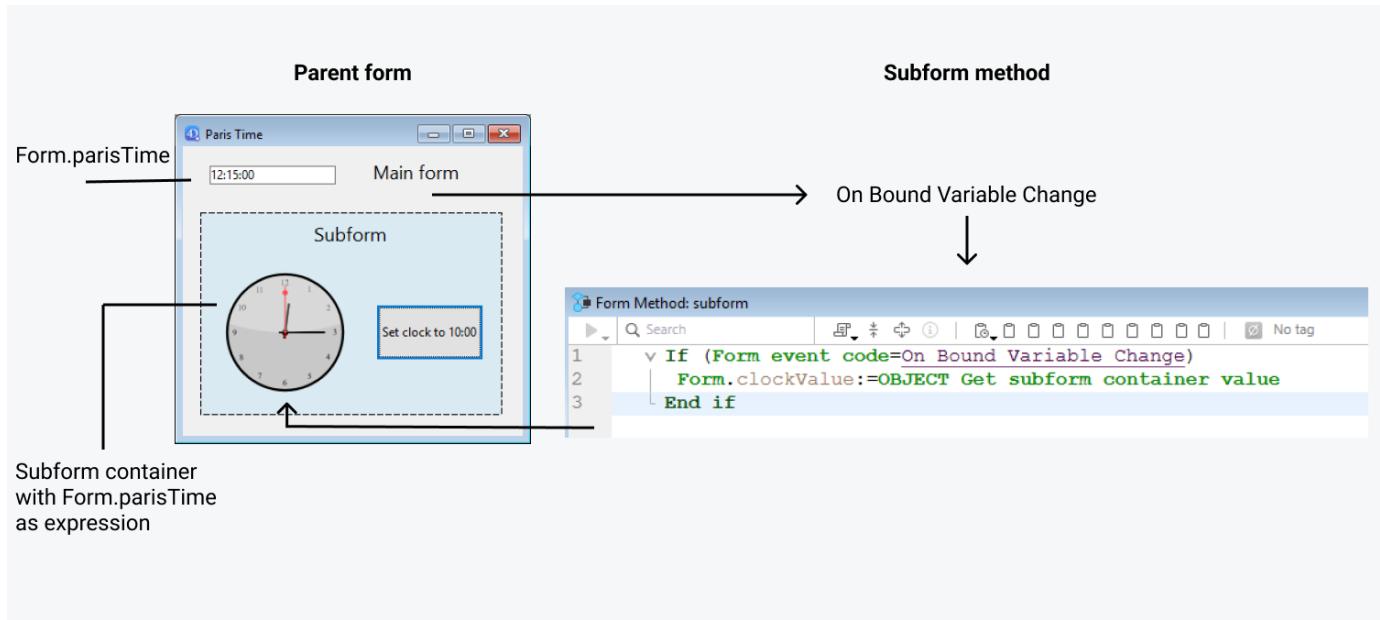
ケース1：親フォームの変数あるいは式の値が更新され、この更新をサブフォームに反映させる必要があります。

この例では、親フォームにおいて Form.parisTime の値がユーザー入力、あるいは動的に（たとえば Current time コマンドで）12:15:00 に変更されました。これは、サブフォームのフォームメソッドで On Bound Variable Change イベントをトリガードします。

以下のコードが実行されます：

```
// サブフォームのフォームメソッド
If (Form event code=On Bound Variable Change)
// 親フォーム内でバインドされた変数あるいは式が変更されました
    Form.clockValue:=OBJECT Get subform container value
    // 親フォームのサブフォームコンテナー値を取得し、ローカルの値を同期させます
End if
```

上のコードは、サブフォームの Form.clockValue の値を更新します：



On Bound Variable Change フォームイベントは以下のときに生成されます:

- 親フォームの変数/式に値が割り当てられたとき（同じ値が再代入された場合でも）で、
- サブフォームが 0 ページまたはカレントフォームページに置かれているとき。

先の例のとおり、式を直接使用するのではなく、親フォームのサブフォームコンテナーの式の値を取得する **OBJECT Get subform container value** コマンドの利用が推奨されます。親フォームに同じサブフォームを複数配置することが可能だからです（たとえば、複数のタイムゾーンを表示するために時計を複数表示するウィンドウ）。

バインドされた変数あるいは式を変更すると、フォームイベントが発生し、親フォームとサブフォームの値を同期させることができます:

- 親フォームのサブフォームコンテナーの変数あるいは式が変更されたことをサブフォーム（のフォームメソッド）に通知するには、サブフォームの **On Bound Variable Change** フォームイベントを使用します。
- 同様に、サブフォーム内で変数あるいは式の値が変更されたことを親フォームのサブフォームコンテナーに通知するには、サブフォームの **On Data Change** フォームイベントを使用します。

親フォームの内容を更新する

ケース2: サブフォームの内容が更新され、その更新を親フォームに反映させる必要があります。

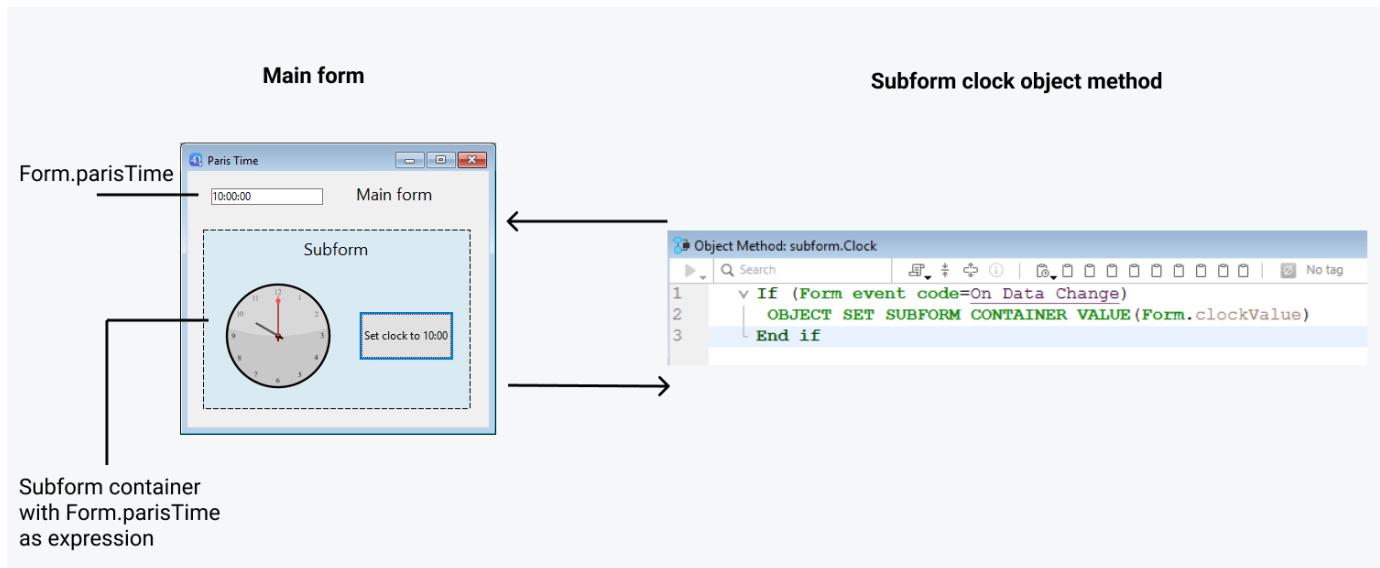
サブフォーム内で、時計オブジェクトに紐づいた **Form.clockValue** 式（時間型）の値がボタンによって変更されます。これにより、時計オブジェクトの内部で **On Data Change** フォームイベント（当該イベントがオブジェクトに対して選択されている必要があります）がトリガーされ、親フォームの **Form.parisTime** 値を更新します。

以下のコードが実行されます:

```

// サブフォームの時計オブジェクトのメソッド
If (Form event code=On Data Change)
// 値が変化したときに
    OBJECT SET SUBFORM CONTAINER VALUE(Form.clockValue)
    // 親フォームのサブフォームコンテナーに値をプッシュします
End if

```



サブフォームの `Form.clockValue` の値が変化するたびに、サブフォームコンテナーの `Form.parisTime` も更新されます。

変数あるいは式の値が複数の場所で設定されている場合、4D は最後にロードされた値を使用します。以下のロード順が適用されます：

- 1 - サブフォームのオブジェクトメソッド
- 2 - サブフォームのフォームメソッド
- 3 - 親フォームのオブジェクトメソッド
- 4 - 親フォームのフォームメソッド

親フォームとサブフォームの同期 (複数値)

デフォルトで、4D は各サブフォームコンテナーに **オブジェクト型** の変数あるいは式を作成します。このオブジェクトの中身は親フォームおよびサブフォームから読み書き可能なため、ローカルなコンテキストにおいて複数の値を共有することができます。

親フォームのサブフォームコンテナーにバインドされたオブジェクトは、サブフォーム内では直接 `Form` コマンドによって返されます。オブジェクトは常に参照によって渡されるため、ユーザーがサブフォーム内でプロパティ値を変更した場合には、その値は自動的にオブジェクト自身に保存され、親フォームでも利用できるようになります。一方、ユーザーによって、あるいはプログラミングによってオブジェクトのプロパティが親フォーム内で変更された場合も、その値はサブフォーム内で自動更新されます。つまり、イベント管理は必要ありません。

たとえば、サブフォームでは、入力は (サブフォームの) `Form` オブジェクトプロパティにバインドされています。

Lastname:	<code>Form.lastname</code>
Firstname:	<code>Form.firstname</code>

親フォームで、このサブフォームを 2回表示するとします。各サブフォームコンテナーは、(親フォームの) `Form` オブジェクトのプロパティである式にバインドされています。

Father	<code>Form.father.lastname</code>	Form or Expression: <code>Form.father</code>
Lastname:	<code>Form.lastname</code>	
Firstname:	<code>Form.firstname</code>	

Mother	<code>Form.mother.lastname</code>	Form or Expression: <code>Form.mother</code>
Lastname:	<code>Form.lastname</code>	
Firstname:	<code>Form.firstname</code>	

Add values

下のボタンは、親フォームの `Form` オブジェクトに `mother` と `father` プロパティを作成します。

```
// Add values ボタンのオブジェクトメソッド
Form.mother:=New object("lastname"; "Hotel"; "firstname"; "Anne")
Form.father:=New object("lastname"; "Golf"; "firstname"; "Félix")
```

フォームを実行してボタンをクリックすると、サブフォームを含め、すべての値が正しく表示されていることがわかります：

Father	
Lastname:	Golf
Firstname:	Félix

Mother	
Lastname:	Hotel
Firstname:	Anne

Add values

同じオブジェクトが使用されているため、親フォームまたはサブフォームで値を変更すると、もう一方のフォームでも値が自動更新されます：

Father	
Lastname:	Wolf
Firstname:	Félix

Mother	
Lastname:	Hotelle
Firstname:	Anne

ポインターの使用 (互換性)

4D v19 R5 以前のバージョンでは、親フォームとサブフォーム間の同期は ポインター を使っておこなわれていました。たとえば、サブフォームオブジェクトを更新するには、以下のコードを呼び出しておこなえます：

```
// サブフォームメソッド
If (Form event code=On Bound Variable Change)
  ptr:=OBJECT Get pointer(0bject subform container)
  clockValue:=ptr->
End if
```

**この方法は互換性のために引き続きサポートされますが、サブフォームに式をバインドすることができないため、廃止予定となります。今後の開発では、この原則は使うべきではありません。すべての場合において、フォームとサブフォームの値を同期させるには、[Form コマンド](#) か [OBJECT Get subform container value](#) と [OBJECT SET SUBFORM CONTAINER VALUE コマンド](#) を使用することが推奨されます。

高度なフォーム間通信プログラム

親フォームとサブフォームインスタンス間の通信では、バインドした変数を通して値を交換する以上のことをおこなう必要がある場合があります。実際、親フォームでおこなわれたアクションに基づきサブフォーム中の変数を更新したり、その逆の処理をしたい場合があるでしょう。先の "動的な時計" タイプのサブフォームの例で言えば、各時計ごとにアラーム時刻を複数設定したい場合が考えられます。

このようなニーズにこたえるため、4Dは以下のメカニズムを実装しています：

- [CALL SUBFORM CONTAINER](#) コマンドを使用してサブフォームからコンテナーオブジェクトを呼び出す、

- EXECUTE METHOD IN SUBFORM コマンドを使用してサブフォームのコンテキストでメソッドを実行する。

GOTO OBJECT はサブフォームから実行されても、親フォーム内にて目的のオブジェクトを検索します。

CALL SUBFORM CONTAINER コマンド

CALL SUBFORM CONTAINER コマンドを使用すると、サブフォームインスタンスからサブフォームコンテナーオブジェクトに **イベント** を送信できます。イベントはコンテナーオブジェクトメソッドで受信されます。(クリックやドラッグ & ドロップなど) サブフォームにより検知されたすべてのイベントの発生元となります。

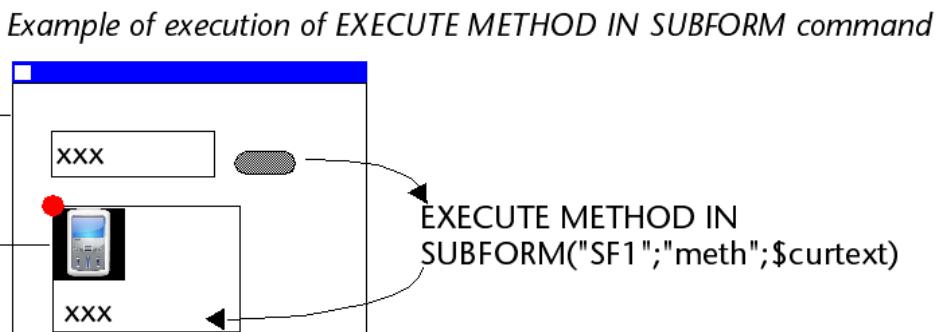
送信するイベントコードに制限はありません (たとえば 20000 や -100 など)。既存のイベントに対応するコード (たとえば **On Validate** に対応する 3) を使用することも、カスタムコードを使用することもできます。前者のケースでは、サブフォームコンテナーのプロパティリストでチェックを入れたイベントのみを使用できます。後者の場合、使用するコードは既存のフォームイベントに対応してはいけません。将来の 4Dバージョンで番号が衝突しないようにするために、負数の使用が推奨されます。

詳細は **CALL SUBFORM CONTAINER** コマンドの説明を参照してください。

EXECUTE METHOD IN SUBFORM コマンド

EXECUTE METHOD IN SUBFORM コマンドを使用すると、親フォームやそのオブジェクトから、サブフォームインスタンスのコンテキストにおけるメソッド実行をリクエストできます。これにより、サブフォームの変数やオブジェクト等にアクセスすることができます。このメソッドは引数も受け取れます。

このメカニズムを図示すると以下のようになります:



詳細は **EXECUTE METHOD IN SUBFORM** コマンドの説明を参照してください。

プロパティ一覧

タイプ - オブジェクト名 - 変数あるいは式 - 式の型 - CSSクラス - ソース - リストフォーム - 詳細フォーム - 選択モード - リスト更新可 - 行をダブルクリック - 空行をダブルクリック - 左 - 上 - 右 - 下 - 幅 - 高さ - 横方向サイズ変更 - 縦方向サイズ変更 - フォーカス可 - 表示状態 - フォーカスの四角を隠す - 横スクロールバー - 縦スクロールバー - 境界線スタイル - 印刷時可変 - メソッド

タブコントロール

タブコントロールは、ユーザーが複数の仮想画面の中から選択することができるオブジェクトを作成します。この画面はタブコントロールオブジェクトにより囲まれています。タブをクリックすることで、各画面にアクセスします。

次の複数ページフォームではタブコントロールオブジェクトが使用されています：

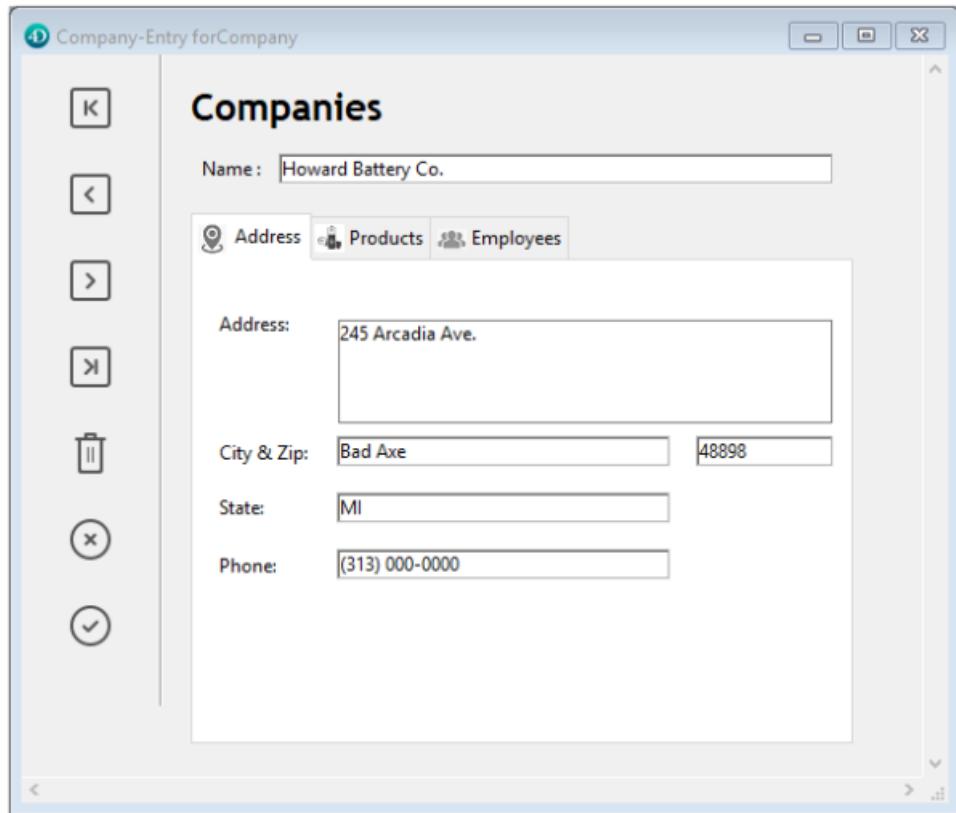
The screenshot shows a Windows-style application window titled "Company-Entry". On the left side, there is a vertical toolbar with several icons: a key icon (K), left arrow, right arrow, forward slash, delete, and checkmark. The main area is titled "Companies". It contains a "Name" field with the value "Howard Battery Co.". Below it is a tabbed panel with three tabs: "Address", "Products", and "Employees". The "Address" tab is selected and active. Inside this tab, there are fields for "Address" (containing "245 Arcadia Ave."), "City & Zip" (containing "Bad Axe" and "48898"), "State" (containing "MI"), and "Phone" (containing "(313) 000-0000").

各画面を移動するには、目的のタブをクリックします。

これらの画面は、マルチページフォームの各ページを表わしたり、またはユーザーがタブがクリックすると変化するオブジェクトを表わすこともできます。タブコントロールをページ移動ツールとして使用する場合、ユーザーがタブをクリックすると `FORM GOTO PAGE` コマンドまたは `gotoPage` 標準アクションを使用します。

タブコントロールの他の利用法は、サブフォームやリストボックスに表示されるデータを制御することです。たとえば、名刺帳はタブコントロールを用いて実現することができます。タブにはひらがなの各文字を表示し、タブコントロールの動作としてはユーザーがクリックした文字と一致するデータをロードします。

各タブにはラベルだけ、またはラベルと小さなアイコンを表示することができます。アイコンが含まれる場合、そのアイコンは各ラベルの左側に表示されます。次の図はアイコンを使用するタブコントロールの例です：



タブコントロールを作成すると、4Dがタブの間隔と配置を管理します。ラベルは配列形式で定義し、アイコンとラベルは階層リスト形式で定義します。

タブコントロールが十分大きく、ラベルとアイコンが設定されたタブをすべて表示できる場合は、その両方が表示されます。タブコントロールの大きさが足らず、ラベルとアイコンを両方とも表示できない場合には、4Dはアイコンだけを表示します。すべてのアイコンが収まりきらない場合、表示される最後のタブの右側にスクロール矢印が置かれます。このスクロール矢印を使用し、アイコンを左右にスクロールできます。

macOSの場合、タブコントロールを標準位置（上）だけでなく、下にも配置することができます。

JSON 例:

```
"myTab": {
    "type": "tab",
    "left": 60,
    "top": 160,
    "width": 100,
    "height": 20,
    "labelsPlacement": "bottom" // タブコントロールの位置
}
```

タブコントロールへのラベルの追加

タブコントロールにラベルを設定するには、次のものを利用できます:

- オブジェクト
- 選択リスト
- 配列

オブジェクトの使用

[コレクション](#) をカプセル化した [オブジェクト](#) をタブコントロールの [データソース](#) として割り当てることができます。このオブジェクトには、次のプロパティが格納されていくことはなりません:

プロパティ	タイプ	説明
values	コレクション	必須 - スカラー値のコレクション。文字列の値のみがサポートされています。無効、空、または未定義の場合、タブコントロールは空になります
index	number	タブコントロールのカレントページのインデックス (0 と <code>collection.length-1</code> の間の値)
currentValue	テキスト	現在選択されている値

この初期化コードはユーザーにフォームを表示する前に実行しなければなりません。

次の例では、タブコントロールの `式` として `Form.tabControl` が定義されています。フォームオブジェクトに `gotoPage` 標準アクション を関連付けることができます。

```
Form.tabControl:=New object
Form.tabControl.values:=New collection("Page 1"; "Page 2"; "Page 3")
Form.tabControl.index:=2 // ページ3 から開始します
```

選択リストの使用

タブコントロールに [選択リスト](#) を関連付けることができます。これにはコレクション (静的リスト)、または jsonリスト ("\$ref") への JSONポインターを使用します。リストエディターにてリスト項目に関連付けられたアイコンはタブコントロールに表示されます。

テキスト配列の使用

フォームの各ページの名前を格納するテキスト配列を作成することができます。このコードはユーザーにフォームを表示する前に実行しなければなりません。たとえば、このコードをタブコントロールのオブジェクトメソッドに置いて、`On Load` イベントが生じたときにこのメソッドを実行します。

```
ARRAY TEXT(arrPages;3)
arrPages{1}:="Name"
arrPages{2}:="Address"
arrPages{3}:="Notes"
```

ページの名前を階層リストに保存し、[LIST TO ARRAY](#) コマンドを使用して値をロードすることも可能です。

Goto page 機能

FORM GOTO PAGE コマンド

タブコントロールのメソッドで `FORM GOTO PAGE` コマンドを使用できます：

```
FORM GOTO PAGE(arrPages)
```

`On Clicked` イベントが発生すると、このコマンドが実行されます。この後 `On Unload` イベントの発生時にこの配列をクリアします。

オブジェクトメソッドの例を次に示します：

```
Case of
:(Form event=On Load)
  LIST TO ARRAY("Tab Labels";arrPages)
:(Form event=On Clicked)
  FORM GOTO PAGE(arrPages)
:(Form event=On Unload)
  CLEAR VARIABLE(arrPages)
End case
```

gotoPage 標準アクション

タブコントロールに `gotoPage` 標準アクション を割り当てることができます。すると、4Dはクリックされたタブコントロールの番号に相当するフォームのページを自動的に表示します。

たとえば、ユーザーが 3番目のタブをクリックすると、4Dはカレントフォームの 3ページ目 (存在する場合) を表示します。

プロパティ一覧

タイプ - オブジェクト名 - 変数あるいは式 - 式の型 - 値を記憶 - CSSクラス - 選択リスト - 左 - 上 - 右 - 下 - 幅 - 高さ - 横方向サイズ変更 - 縦方向サイズ変更 - 表示状態 - タブコントロールの位置 - フォント - フォントサイズ - 太字 - イタリック - 下線 - ヘルプTips - 標準アクション

テキスト

テキストオブジェクトを使って、指示・タイトル・ラベルなどの静的（スタティック）なテキストをフォーム上に表示することができます。これらのテキストは、参照を含むことで動的にもなります。詳細については [スタティックテキスト中で参照を使用する](#) を参照ください。

JSON 例：

```
"myText": {  
    "type": "text",  
    "text": "Hello World!",  
    "textAlign": "center",  
    "left": 60,  
    "top": 160,  
    "width": 100,  
    "height": 20,  
    "stroke": "#ff0000"      // テキストカラー  
    "fontWeight": "bold"  
}
```

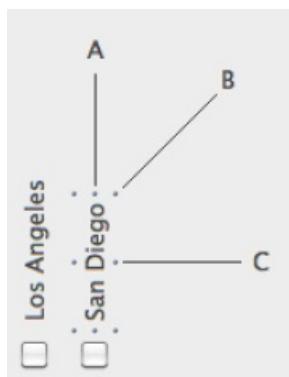
回転

4D では、フォーム内のテキストエリアを **回転** させることができます。

	New York	Chicago	Los Angeles	San Diego
Alpha	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Bravo	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Charlie	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Delta	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Echo	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

このプロパティは `OBJECT SET TEXT ORIENTATION` ランゲージコマンドによっても設定することができます。

テキストが回転された後でも、サイズや位置などすべてのプロパティを変更することが可能です。テキストエリアの高さと幅は、回転の方向に依らないという点に注意してください：



- オブジェクトが A 方向にリサイズされるとき、変更されるのは **幅** です。
- オブジェクトが C 方向にリサイズされるとき、変更されるのは **高さ** です。
- オブジェクトが B 方向にリサイズされるとき、**幅** と **高さ** の両方が同時に変更されます。

プロパティ一覧

タイプ - オブジェクト名 - タイトル - CSSクラス - 左 - 上 - 右 - 下 - 幅 - 高さ - 横方向サイズ変更 - 縦方向サイズ変更 - 表示状態 - 塗りカラー - 境界線スタイル - フォント - フォントサイズ - 太字 - イタリック - 下線 - フォントカラー - 方向 - 横揃え

Webエリア

Webエリアは、静的および動的な HTML ページ、ファイル、ピクチャー、JavaScript などの様々な Web コンテンツをフォームの中で表示することができるオブジェクトです。Web エリアの描画エンジンは、アプリケーションの実行プラットフォームおよび [埋め込み Web レンダリングエンジンを使用](#) オプションの設定状態により異なります。

同じフォーム内に複数の Web エリアを配置できます。しかしながら、Web エリアの挿入には [いくつかの制約](#) がつく事に注意して下さい。

いくつかの専用の [標準アクション](#)、多数の [ランゲージコマンド](#)、そして汎用および専用の [フォームイベント](#) を使用して、Web エリアの動作を制御することができます。特別な変数を使用して、エリアと 4D 環境間で情報を交換することも可能です。

特有のプロパティ

割り当てられる変数

Web エリアには 2 つの特別な変数が自動で割り当てられます:

- [URL](#) -- Web エリアに表示されている URL の管理に使用します。
- [進捗状況変数](#) -- Web エリアにロード中のページのパーセンテージを知るために使用します。

4D v19 R5 以降、[Windows システムレンダリングエンジン](#) を使用する Web エリアでは、進捗状況変数が更新されません。

Web レンダリングエンジン

Web エリアでは、[2 つの描画エンジン](#) うちから使用するものを選択することができます。

"埋め込み Web レンダリングエンジンを使用" プロパティを選択している場合、"4D メソッドコードを許可" プロパティが選択可能になり、また、macOS と Windows 上の動作が同様であるようにできます。Web エリアがインターネットに接続されている場合には、最新のセキュリティアップデートの恩恵を受けられるため、システムレンダリングエンジンを選択することが推奨されます。

4D メソッドコードを許可

[4D メソッドコードを許可](#) プロパティを選択している場合、Web エリアから 4D メソッドを呼び出すことができます。

この機能は Web エリアが [埋め込み Web レンダリングエンジンを使用](#) している場合に限り、使用可能です。

\$4d オブジェクトの使用

[4D の埋め込み Web レンダリングエンジン](#) は、\$4d という JavaScript オブジェクトをエリアに提供します。\$4d オブジェクトと " ." (ドット) オブジェクト記法を使用することによって、任意の 4D プロジェクトメソッドを呼び出すことができます。

たとえば、[HelloWorld](#) という 4D メソッドを呼び出す場合には、以下の宣言を実行します:

```
$4d.HelloWorld();
```

JavaScript は大文字小文字を区別するため、オブジェクトの名前は \$4d (d は小文字) であることに注意が必要です。

4D メソッドへの呼び出しのシンタックスは以下のようになります:

```
$4d.4DMethodName(param1,paramN,function(result){})
```

- param1...paramN : 4Dメソッドに対して必要なだけ引数を渡すことができます。これらの引数は、JavaScript にサポートされている型であればどんなものでも渡せます（文字列、数値、配列、オブジェクト）。
- function(result) : 最後の引数として渡される関数です。この "コールバック" 関数は、4Dメソッドが実行を終えると同時に呼び出されます。この関数は result 引数を受け取ります：
- result : "\$0" 式に返される、4Dメソッド実行の戻り値です。戻り値は JavaScript でサポートされている型（文字列、数値、配列、オブジェクト）のいずれかになります。C_OBJECT コマンドを使用して、オブジェクトを返すことができます。

デフォルトとして、4Dは UTF-8 文字コードで動作しています。（アクセントが付いた文字などの）拡張文字を含むテキストを返す場合には、Webエリアで表示されるページの文字コードが UTF-8 に宣言されていることを確認してください。文字コードが UTF-8 でない場合、文字が正しく表示されない可能性があります。この場合、以下の 1行を HTMLページに追加して文字コードを宣言してください：

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

例題 1

today という名の 4Dプロジェクトメソッドがあり、そのメソッドは引数を受け付けず、カレントの日付を文字列として返す場合について考えてみます。

today メソッドの 4Dコードです：

```
C_TEXT($0)
$0:=String(Current date;System date long)
```

Webエリアでは、4Dメソッドは以下のシンタックスで呼び出し可能です：

```
$4d.today()
```

この 4Dメソッドは引数を受け取りませんが、メソッドの実行後に \$0 の値を、4Dによって呼び出されるコールバック関数へと返します。Webエリアによってロードされた HTMLページ内にこの日付を表示します。

HTMLページのコードです：

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<script type="text/javascript">
$4d.today(function(dollarZero)
{
    var curDate = dollarZero;
    document.getElementById("mydiv").innerHTML=curDate;
});
</script>
</head>
<body>今日は： <div id="mydiv"></div>
</body>
</html>
```

例題 2

calcSum という 4Dプロジェクトメソッドがあり、そのメソッドが(``\$1...\$n) という引数を受け取り、その合計を \$0 に返すという場合について考えます。

calcSum メソッドの 4Dコードです：

```

C_REAL(${1}) // n個の実数型引数を受け取ります
C_REAL($0) // 実数の値を返します
C_LONGINT($i;$n)
$n:=Count parameters
For($i;1;$n)
    $0:=$0+$i
End for

```

Webエリア内で実行される JavaScript コードです:

```

$4d.calcSum(33, 45, 75, 102.5, 7, function(dollarZero)
{
    var result = dollarZero // 結果は 262.5 です
});

```

標準アクション

Webエリアを自動で管理するために、4つの特別な自動アクション（`openBackURL`、`openForwardURL`、`refreshCurrentURL`、そして`stopLoadingURL`）を使用できます。ボタンやメニュー命令にこれらのアクションを割り当てることで、基本の Webインターフェースを素早く実装できます。これらのアクションについては [標準アクション](#) で説明しています。

フォームイベント

特定のフォームイベントは、Webエリアをプログラミングで管理することを目的としています。すなわち、リンクの起動に関連しています:

- [On Begin URL Loading](#)
- [On URL Resource Loading](#)
- [On End URL Loading](#)
- [On URL Loading Error](#)
- [On URL Filtering](#)
- [On Open External Link](#)
- [On Window Opening Denied](#)

更に、Webエリアは以下の汎用フォームイベントをサポートしています:

- [On Load](#)
- [On Unload](#)
- [On Getting Focus](#)
- [On Losing Focus](#)

Webエリアのルール

ユーザーインターフェース

フォームが実行されると、他のフォームエリアとの対話を可能にする、標準のブラウザインターフェース機能が Web エリア内で利用可能になります。

- 編集メニュー命令: Webエリアにフォーカスがあるとき、編集 メニュー命令を使用してコピー・ペースト、すべてを選択などのアクションを選択に応じて実行できます。
- コンテキストメニュー: Webエリアで、システム標準の [コンテキストメニュー](#) を使用できます。コンテキストメニューの表示は、`WA SET PREFERENCE` コマンドを使用することで管理可能です。
- ドラッグ & ドロップ: 4D のオブジェクトプロパティに基づき、ユーザーは Webエリア内で、または Webエリアと 4Dフォームオブジェクト間で、テキストやピクチャ、ドキュメントをドラッグ & ドロップできます。セキュリティ上の理由から、ファイルまたは URL のドラッグ & ドロップによって Webエリアのコンテンツを変更することは、デフォルトで禁止されています。この場合、カーソルは "禁止" アイコン  を表示します。"ドロップ" アイコンを表示し、[On Window Opening Denied](#) イベントを発生させるには、`WA SET PREFERENCE(*; "warea"; WA enable URL drop; True)` 文を使い

ます。このイベントでは、WA OPEN URL コマンドを呼び出したり、ユーザードロップに対する URL 変数 を設定することができます。

上記のドラッグ & ドロップ機能は、macOS のシステムレンダリングエンジンを使用している Webエリアではサポートされません。

サブフォーム

ウィンドウの再描画機構に関わる理由から、サブフォームに Webエリアを挿入する場合には以下の制約がつきます：

- サブフォームをスクロール可能にしてはいけません。
- Webエリアのサイズがサブフォームのサイズを超えてはいけません。

他のフォームオブジェクトの上や下に Webエリアを重ねることはサポートされていません。

Webエリアと Webサーバーのコンフリクト (Windows)

Windowsにおいては、Webエリアから、同じ 4D アプリケーションで起動中の Webサーバーへのアクセスはお勧めできません。これをおこなうとコンフリクトが発生し、アプリケーションがフリーズすることがあります。もちろん、リモートの 4D から 4D Server の Webサーバーにアクセスすることはできます。自身の Webサーバーにアクセスできないということです。

プロトコルの挿入 (macOS)

macOS 上の Webエリアで、プログラムにより処理される URL は、プロトコルで開始されなければなりません。つまり、"www.mysite.com" ではなく、"<http://www.mysite.com>" 文字列を渡さなければならないということです。

Webインスペクターへのアクセス

オフスクリーンの Webエリアや、フォームの Web エリア内において、Webインスペクターを見たり使用したりすることができます。Webインスペクターは、埋め込み Webエンジンによって提供されているデバッガーです。Webページの情報の、コードとフローを解析します。

Webインスペクターを表示させるには、WA OPEN WEB INSPECTOR コマンドを実行するか、Webエリアのコンテキストメニューを使用します。

- WA OPEN WEB INSPECTOR コマンドの実行

このコマンドはスクリーン上 (フォームオブジェクト) の、またはオフスクリーンの Webエリアに対して直接使用することができます。

- Webエリアコンテキストメニューの使用

この機能はオンスクリーンの Webエリアでのみ使用することができ、以下の条件を満たしている必要があります：

- エリアに対して コンテキストメニュー が有効化されている。
- インスペクターの使用が、以下の宣言を用いて明示的に有効化されている：

```
WA SET PREFERENCE(*;"WA";WA enable Web inspector;True)
```

Windows のシステムレンダリングエンジン の場合にこの環境設定を変更すると、変更を反映するのにエリア内でのナビゲーション操作 (たとえば、ページの更新) が必要です。

詳細は WA SET PREFERENCE コマンドの説明を参照してください。

上記のとおり設定を完了すると、エリア内のコンテキストメニュー内に 要素を調査 という新しいオプションが追加されているはずです：この項目を選択すると、Webインスペクターウィンドウが表示されます。

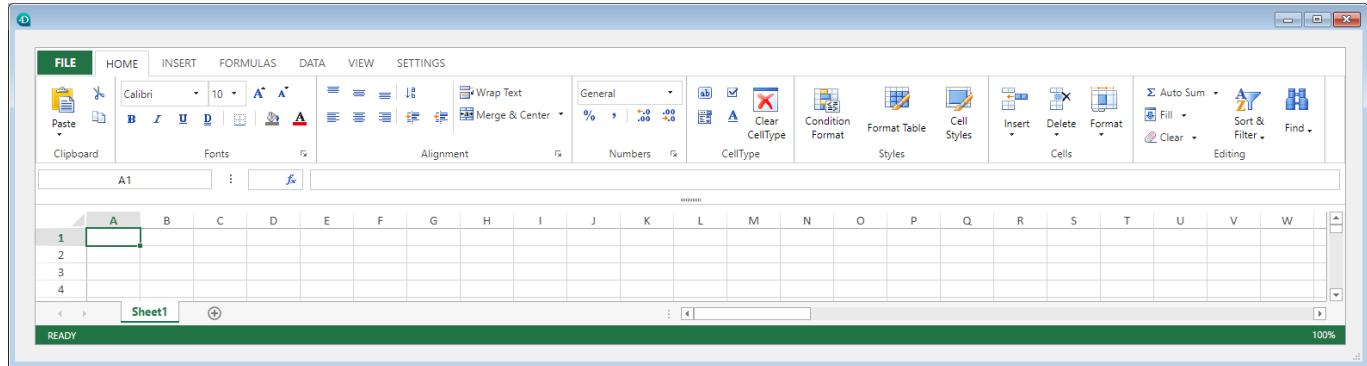
このデバッガーの機能の詳細に関しては、Webレンダリングエンジンにより提供されているドキュメントを参照してください。

プロパティ一覧

タイプ - オブジェクト名 - 変数あるいは式 - CSSクラス - 左 - 上 - 右 - 下 - 幅 - 高さ - 横方向サイズ変更 - 縦方向サイズ変更 - コンテキストメニュー - 表示状態 - 境界線スタイル - メソッド - 進捗状況変数 - URL - 埋め込み Webレンダリングエンジンを使用

4D View Pro area

4D View Pro を使用すると、4Dフォーム内にスプレッドシートエリアを挿入・表示できるようになります。スプレッドシートとは、セルのグリッドを格納したアプリケーションのことで、これらのセルに情報を入力したり、計算を実行させたり、あるいはピクチャーを表示したりすることができます。



フォーム内で 4D View Pro エリアを使用すると、スプレッドシートドキュメントを読み込んだり書き出したりすることができます。

4D View Pro エリアの使用

4D View Pro についての詳細は [4D View Pro](#) の章を参照ください。

プロパティ一覧

タイプ - オブジェクト名 - CSSクラス - 左 - 右 - 下 - 幅 - 高さ - 横方向サイズ変更 - 縦方向サイズ変更 - 表示状態 - ユーザーインターフェース
- フォーミュラバーを表示 - 境界線スタイル - メソッド

4D Write Pro area

4D Write Pro は、4Dユーザーに対して、4Dアプリケーションに完全に統合した高度なワードプロセスツールを提供します。4D Write Proを使用すれば、プリフォーマットされた Eメールや文章に画像、スキャン済みの署名、フォーマットされたテキストやダイナミック変数用のプレースホルダーなどを含めることができます。また、請求書やレポートを動的に作成し、フォーマット済みのテキストや画像を含めることができます。



4D Write Pro エリアの使用

4D Write Pro についての詳細は [4D Write Pro リファレンス マニュアル](#)を参照ください。

プロパティ一覧

タイプ - オブジェクト名 - CSSクラス - 左 - 右 - 下 - 幅 - 高さ - 横方向サイズ変更 - 縦方向サイズ変更 - 入力可 - フォーカス可 - キーボード
レイアウト - 自動スペルチェック - コンテキストメニュー - 選択を常に表示 - 表示状態 - フォーカスの四角を隠す - 横スクロールバー - 縦スクロール
バー - 解像度 - 拡大 - ビューモード - ページフレームを表示 - 参照を表示 - ヘッダーを表示 - フッターを表示 - 背景を表示 - 非表示文字を表示
- HTML WYSIWYG 表示 - 水平ルーラーを表示 - 垂直ルーラーを表示 - 境界線スタイル - 印刷時可変 - メソッド - ドラッグ有効 - ドロップ有
効

JSON プロパティリスト

この一覧は、すべてのオブジェクトプロパティを JSON名順に並べてまとめたものです。プロパティ名をクリックすると、詳細説明に移動します。

"フォームオブジェクトプロパティ" の章では、プロパティリストにおけるテーマと名称で各プロパティを紹介しています。

a - b - c - d - e - f - g - h - i - j - k - l - m - n - p - r - s - t - u - v - w - z

プロパティ	説明	とりうる値
a		
action	実行される一般的な動作	有効な標準アクション名
allowFontColorPicker	オブジェクト属性を変更するためのフォントあるいはカラーピッカーの表示を許可します。	true, false (デフォルト)
alternateFill	奇数番の行/列に使用するための異なる背景色を設定することができます。	任意の CSS値; "transparent", "automatic", "automaticAlternate"
automaticInsertion	オブジェクトに割り当てられた選択リストに無い値をユーザーが入力した場合に、その値をリストに自動的に追加します。	true, false
b		
booleanFormat	二つの値しか取れないことを指定します。	true, false
borderRadius	角の丸い四角形の角の半径	最小値: 0
borderStyle	リストボックスの境界線のスタイルを設定します。	"system", "none", "solid", "dotted", "raised", "sunken", "double"
bottom	フォーム上のオブジェクトの下の座標	最小値: 0
c		
choiceList	オブジェクトに割り当てられた選択肢のリスト	選択肢のリスト
class	css ファイルにてクラスセレクターとして使用される、(複数の場合は半角スペース区切りの) クラス名のリスト。	クラス名のリスト
columnCount	列数	最小値: 1
columns	リストボックス列のコレクション	定義された列プロパティを格納した列オブジェクトのコレクション
contextMenu	選択されたエリア内でユーザーに標準のコンテキストメニューを提供します。	"automatic", "none"
continuousExecution	ユーザーによるコントロールのトラッキング中に、オブジェクトメソッドを実行するかどうかを指定します。	true, false
controlType	リストボックスのセル内で値がどのように表示されるかを指定します。	"input", "checkbox" (ブール/数値型カラムのみ), "automatic", "popup" (ブール型カラムのみ)
currentItemSource	リストボックス内で最後に選択された	オブジェクト型の式

項目	説明	とりうる値
<code>currentItemPositionSource</code>	リストボックス内で最後に選択された項目の位置	数値型の式
<code>customBackgroundPicture</code>	ボタンの背景に描画されるピクチャーを設定します。	POSIX シンタックスの相対パス。style プロパティの "custom" オプションと併用する必要があります。
<code>customBorderX</code>	オブジェクト内部の水平方向のマージンのサイズを設定します(ピクセル単位)。style プロパティの "custom" オプションと併用する必要があります。	最小値: 0
<code>customBorderY</code>	オブジェクト内部の垂直方向のマージンのサイズを設定します(ピクセル単位)。style プロパティの "custom" オプションと併用する必要があります。	最小値: 0
<code>customOffset</code>	カスタムのオフセット値をピクセル単位で設定します。style プロパティの "custom" オプションと併用する必要があります。	最小値: 0
<code>customProperties</code>	高度なプロパティ(あれば)	JSON 文字列、または base64 エンコードの文字列
d		
<code>dataSource</code> (オブジェクト) <code>dataSource</code> (サブフォーム) <code>dataSource</code> (配列リストボックス) <code>dataSource</code> (コレクションまたはエンティティセレクションリストボックス) <code>dataSource</code> (リストボックス列) <code>dataSource</code> (階層リストボックス)	データのソースを指定します。	4D変数、フィールド名、あるいは任意のランゲージ式
<code>dataSourceTypeHint</code> (オブジェクト) <code>dataSourceTypeHint</code> (リストボックス列、ドロップダウンリスト)	変数の型を示します。	"integer", "boolean", "number", "picture", "text", "date", "time", "arrayText", "arrayDate", "arrayTime", "arrayNumber", "collection", "object", "undefined"
<code>dateFormat</code>	表示/印刷時に日付をどのように表示かを管理します。4D にビルトインされているフォーマットから選択されなければなりません。	"systemShort", "systemMedium", "systemLong", "iso8601", "rfc822", "short", "shortCentury", "abbreviated", "long", "blankIfNull" (他の値と組み合わせることができます)
<code>defaultButton</code>	ボタンの見た目を変更することで、ユーザーに対してこのボタンが推奨される選択であることを示します。	true, false
<code>defaultValue</code>	入力オブジェクトにデフォルトで表示する値やスタンプを指定します。	文字列、または "#D", "#H", "#N"
<code>deletableInList</code>	リストサブフォーム内でユーザーがサブレコードを削除できるかどうかを指定します。	true, false
<code>detailForm</code> (リストボックス) <code>detailForm</code> (サブフォーム)	詳細フォームをリストサブフォームに関連づけます。	テーブルまたはプロジェクトフォームの名前(文字列)、フォームを定義する .json ファイルへの POSIX パス(文字列)、またはフォームを定義するオブジェクト
<code>display</code>	フォーム上にオブジェクトを描画するかどうかを指定します。	true, false
<code>doubleClickInEmptyAreaAction</code>	リストサブフォームの空行がダブルクリックされた際に実行されるアクションを指定します。	"addSubrecord", 何もしない場合は ""

プロパティ	説明	値
<code>clickInRowAction</code> (リストボックス)	リストをダブルクリックしたときに実行するアクション	<code>"showSubRecord", "displaySubrecord"</code>
<code>doubleClickInRowAction</code> (サブフォーム)		
<code>dpi</code>	4D Write Pro エリアの画面解像度	0 = 自動, 72, 96
<code>dragging</code>	ドラッグ機能を有効化します。	"none", "custom", "automatic" (リストとリストボックスを除く)
<code>dropping</code>	ドロップ機能を有効化します。	"none", "custom", "automatic" (リストとリストボックスを除く)
<code>e</code>		
<code>enterable</code>	ユーザーがオブジェクトに値を入力できるかどうかを指定します。	true, false
<code>enterableInList</code>	リストサブフォームにおいて、ユーザーがレコードデータを直接編集できるかどうかを指定します。	true, false
<code>entryFilter</code>	オブジェクトあるいはカラムのセルに入力フィルターを割り当てます。このプロパティは enterable プロパティが有効化されていない時には利用できません。	入力フィルターを定義するテキスト
<code>events</code>	オブジェクトまたはフォームについて選択されているイベントのリスト	イベント名のコレクション。例: <code>["onClick", "onDataChange"...]</code> .
<code>excludedList</code>	除外リストを使用すると、当該リストの項目は入力できなくなります。	除外する値のリスト
<code>f</code>		
<code>fill</code>	オブジェクトの背景色を設定します。	任意の css 値; "transparent"; "automatic"
<code>focusable</code>	オブジェクトがフォーカスを得られるか(つまり、キーボードなどを使用してアクティブ化できるか)を指定します。	true, false
<code>fontFamily</code>	オブジェクト内で使用されるフォントを指定します。	CSS フォントファミリー名
<code>fontSize</code>	フォントテーマが選択されていない場合に、フォントサイズを指定します(ポイント単位)。	最小値: 0
<code>fontStyle</code>	選択テキストの線を右斜めに傾けます。	"normal", "italic"
<code>fontTheme</code>	自動スタイルを適用します。	"normal", "main", "additional"
<code>fontWeight</code>	選択テキストの線を太くし、濃く見えるようにします。	"normal", "bold"
<code>footerHeight</code>	フッターの高さを指定します。	パターン: <code>^(¥¥d+)(px em)?\$</code> (正の小数 + px/em)
<code>frameDelay</code>	このモードを使用すると、一定のスピードで (tick 単位) ピクチャーボタンの内容が繰り返し表示されます。	最小値: 0
<code>g</code>		
<code>graduationStep</code>	目盛の表示単位です。	最小値: 0
<code>h</code>		

プロジェクト	既存ボックス列のヘッダーを定義します。	値 "name", "icon", "dataSource", "fontWeight", "fontStyle", "tooltip" のプロパティを格納するオブジェクト
headerHeight	フッターの高さを指定します。	パターン: ^(¥¥d+)(px em)?\$ (正の小数 + px/em)
height	オブジェクトの縦のサイズを指定します。	最小値: 0
hideExtraBlankRows	追加の空白行を非表示にします。	true, false
hideFocusRing	オブジェクトにフォーカスがあるときに選択状態を表す強調用の四角形を非表示にします。	true, false
hideSystemHighlight	リストボックス内の選択レコードのハイライトを非表示にします。	true, false
highlightSet	string	セットの名称
horizontalLineStroke	リストボックス内の横線の色を指定します (デフォルトはグレー)。	任意の css 値; "transparent"; "automatic"
i		
icon	ボタン、チェックボックス、ラジオボタン、リストボックスヘッダーに使用するピクチャーのパス名	POSIX シンタックスの相対パス、またはファイルシステムパス
iconFrames	ピクチャー内で表示される状態の数を設定します。	最小値: 1
iconPlacement	フォームオブジェクトに対するアイコンの配置を指定します。	"none", "left", "right"
k		
keyboardDialect	入力オブジェクトに特定のキーボードレイアウトを割り当てます。	入力ロケールを指定する文字列 (例: "ar-ma")
l		
labels	タブコントロールラベルに使用する値のリスト	例: "a", "b", "c", ...
labelsPlacement (オブジェクト) labelsPlacement (タブコントロール)	ラベルが表示される際の位置です。	"none", "top", "bottom", "left", "right"
layoutMode	フォームエリア内の 4D Write Pro ドキュメントの表示モード	"page", "draft", "embedded"
left	フォーム上のオブジェクトの左の座標	最小値: 0
list (choiceList 参照)	階層リストに割り当てられた選択肢のリスト	選択肢のリスト
listboxType	リストボックスの種類を指定します。	"array", "currentSelection", "namedSelection", "collection"
listForm	サブフォームで使用するリストフォーム	テーブルまたはプロジェクトフォームの名前 (文字列), フォームを定義する .json ファイルへの POSIX パス (文字列), またはフォームを定義するオブジェクト
lockedColumnCount	リストボックスの左側に常に表示される列の数	最小値: 0
loopBackToFirstFrame	ピクチャーを連続的に表示し続けます。	true, false

m	プロパティ	説明	とりうる値
	<code>max</code>	最大値。時間型のステッパーの場合、値は秒を表します。日付型のステッパーでは、最小および最大プロパティは無視されます。	最小値: 0 (数値型の場合)
	<code>maxWidth</code>	リストボックス列の最大幅 (ピクセル単位)	最小値: 0
	<code>metaSource</code>	スタイルや、選択の可否設定を格納するメタ情報式	オブジェクト型の式
	<code>method</code>	プロジェクトメソッド名	存在するプロジェクトメソッドの名前
	<code>methodsAccessibility</code>	Webエリアから呼び出せる 4Dメソッドを指定します。	"none" (デフォルト), "all"
	<code>min</code>	最小値。時間型のステッパーの場合、値は秒を表します。日付型のステッパーでは、最小および最大プロパティは無視されます。	最小値: 0 (数値型の場合)
	<code>minWidth</code>	リストボックス列の最小幅 (ピクセル単位)	最小値: 0
	<code>movableRows</code>	ランタイムにおける行の移動を許可します。	true, false
	<code>multiline</code>	複数行のテキストの扱いを指定します。	"yes", "no", "automatic"
n			
	<code>name</code>	フォームオブジェクトの名前。(フォーム自身については任意)	既存オブジェクトによって使用されていない名称
	<code>numberFormat</code>	表示や印刷時に文字フィールドや変数にデータを表示する方法を制御します。	数値 (必要に応じて小数点およびマイナス記号を含む)
p			
	<code>picture</code>	ピクチャー ボタン、ピクチャーポップアップメニュー、スタティックピクチャーに使うピクチャーのパス名	POSIXシンタックスの相対パスまたはファイルシステムパス、ピクチャー変数の場合は "var:<variableName>"
	<code>pictureFormat</code> (入力オブジェクト、リストボックス列またはフッター) <code>pictureFormat</code> (スタティックピクチャー)	表示あるいは印刷される際のピクチャーの表示方法を制御します。	"truncatedTopLeft", "scaled", "truncatedCenter", "tiled", "proportionalTopLeft" (スタティックピクチャーを除く), "proportionalCenter" (スタティックピクチャーを除く)
	<code>placeholder</code>	データソース値が空のときに表示される半透明のテキスト	表示する半透明のテキスト
	<code>pluginAreaKind</code>	プラグインが提供する外部エリアの名称	プラグインの外部エリア名
	<code>popupPlacement</code>	ボタン内に逆三角形として表われるシンボルを表示し、ポップアップメニューが付属することを示します。	"None", "Linked", "Separated"
	<code>printFrame</code>	レコードの中身に応じてサイズが変化しうるオブジェクトの印刷モード	"fixed", "variable", (サブフォームのみ) "fixedMultiple"
	<code>progressSource</code>	Webエリアに表示されるページのロードされたパーセンテージを表す 0 から 100 までの値 この変数は 4D が自動で更新します。手動で変更することはできません。	最小値: 0

プロパティ	説明	とりうる値
<code>radioGroup</code>	複数のラジオボタンを連動させるためのプロパティです。同じラジオグループに属している複数のラジオボタンは、一度にその内の一つのみを選択することができます。	ラジオグループ名
<code>requiredList</code>	有効な入力値のリストを指定します。	有効な入力値のリスト
<code>resizable</code>	ユーザーによるオブジェクトサイズの変更が可能かを指定します。	"true", "false"
<code>resizingMode</code>	リストボックス列を自動リサイズするかを指定します。	"rightToLeft", "legacy"
<code>right</code>	フォーム上のオブジェクトの右の座標	最小値: 0
<code>rowControlSource</code>	リストボックス行の表示を管理するための 4D配列	配列
<code>rowCount</code>	列数を指定します。	最小値: 1
<code>rowFillSource</code> (配列リストボックス) <code>rowFillSource</code> (セレクションまたはコレクションリストボックス)	リストボックスの各行にカスタムの背景色を適用するための配列名または式	配列名または式
<code>rowHeight</code>	リストボックス行の高さを設定します。	"em" または "px" (デフォルト) 単位の css値
<code>rowHeightAuto</code>	boolean	"true", "false"
<code>rowHeightAutoMax</code>	リストボックス行の高さの最大値を設定します。	"em" または "px" (デフォルト) 単位の css値 最小値: 0
<code>rowHeightAutoMin</code>	リストボックスの行の高さの最小値を設定します。	"em" または "px" (デフォルト) 単位の css値 最小値: 0
<code>rowHeightSource</code>	リストボックスの各行の高さを指定する配列	4D 配列変数の名前
<code>rowStrokeSource</code> (配列リストボックス) <code>rowStrokeSource</code> (セレクションまたはコレクション/エンティティセレクションリストボックス)	リストボックスの各行にカスタマイズしたフォントカラーを適用するための配列名または式	配列名または式
<code>rowStyleSource</code> (配列リストボックス) <code>rowStyleSource</code> (セレクションまたはコレクション/エンティティセレクションリストボックス)	リストボックスの各行にカスタマイズしたスタイルを適用するための配列名または式	配列名または式
S		
<code>saveAs</code> (リストボックス列) <code>saveAs</code> (ドロップダウンリスト)	フォームオブジェクトに関連付けられたフィールドまたは変数に保存する値の種類	"value", "reference"
<code>scrollbarHorizontal</code>	表示エリアを左右に移動できるようにするツールです。	"visible", "hidden", "automatic"
<code>scrollbarVertical</code>	表示エリアを上下に移動できるようにするツールです。	"visible", "hidden", "automatic"
<code>selectedItemsSource</code>	リストボックス内で選択されている項目のコレクション	コレクション式
<code>selectionMode</code> (階層リスト) <code>selectionMode</code> (リストボックス)	ユーザーがレコードを複数選択できるかを指定します。	"multiple", "single", "none"

<code>selectionMode</code> (ソヘイハシノヘイ) <code>SelectionMode</code> (サブフォーム)	説明	とりうる値
<code>shortcutAccel</code>	使用するシステム (Windows あるいは macOS) を指定します。	true, false
<code>shortcutAlt</code>	ショートカットに Altキーを使用します。	true, false
<code>shortcutCommand</code>	ショートカットに Commandキーを使用します (macOS)。	true, false
<code>shortcutControl</code>	ショートカットに Controlキーを使用します (Windows)。	true, false
<code>shortcutKey</code>	ショートカットに使用する、特別な意味を持つキーの文字あるいは名前	"[F1]" -> "[F15]", "[Return]", "[Enter]", "[Backspace]", "[Tab]", "[Esc]", "[Del]", "[Home]", "[End]", "[Help]", "[Page up]", "[Page down]", "[left arrow]", "[right arrow]", "[up arrow]", "[down arrow]"
<code>shortcutShift</code>	ショートカットに Shiftキーを使用します。	true, false
<code>showFooters</code>	列のフッターを表示/非表示にします。	true, false
<code>showGraduations</code>	ラベルの隣に目盛を表示、または非表示にします。	true, false
<code>showHeaders</code>	列のヘッダーを表示/非表示にします。	true, false
<code>showHiddenChars</code>	非表示の文字を表示/非表示にします。	true, false
<code>showHorizontalRuler</code>	ドキュメントビューがページビューモードの場合に、水平ルーラーを表示/非表示にします。	true, false
<code>showHTMLWysiwyg</code>	HTML WYSIWYG 表示を有効/無効にします。	true, false
<code>showPageFrames</code>	ドキュメントビューがページビューモードの場合に、ページフレームを表示/非表示にします。	true, false
<code>showReferences</code>	ドキュメントに 参照 として挿入された 4D式をすべて表示します。	true, false
<code>showSelection</code>	オブジェクト中で選択した文字列の反転状態が、フォーカスを失った後も表示されるようになります。	true, false
<code>showVerticalRuler</code>	ドキュメントビューがページビューモードの場合に、垂直ルーラーを表示/非表示にします。	true, false
<code>singleClickEdit</code>	編集モードへの直接移行を可能にします。	true, false
<code>sizingX</code>	ユーザーがフォームの幅をサイズ変更したときの、オブジェクトの挙動を指定します。	"grow", "move", "fixed"
<code>sizingY</code>	ユーザーがフォームの高さをサイズ変更したときの、オブジェクトの挙動を指定します。	"grow", "move", "fixed"
<code>sortable</code>	ヘッダーのクリックによる列データの並べ替えを有する。.キヌ	true, false

プロパティ	説明	とりうる値
<code>spellcheck</code>	オブジェクトの自動スペルチェックを有効にします	true, false
<code>splitterMode</code>	プロパティを適用するとスプリッターオブジェクトは"ブッシャー"になり、そのオブジェクトの右側(垂直スプリッター)または下側(水平スプリッター)にある他のオブジェクトは、スプリッターと一緒に押し出されて移動します。	"grow", "move", "fixed"
<code>startPoint</code>	線の始点を定義します(JSON文法でのみ利用可能)	"bottomLeft", "topLeft"
<code>staticColumnCount</code>	実行時にドラッグで移動できない列の数を指定します。	最小値: 0
<code>step</code>	使用時に各値の間にあけることができる最小の間隔です。時間型のステッパーの場合、このプロパティは秒を表します。日付型のステッパーでは日数を表します。	最小値: 1
<code>storeDefaultStyle</code>	変更がおこなわれていなくても、テキストとともにスタイルタグを格納します。	true, false
<code>stroke</code> (テキスト) <code>stroke</code> (線) <code>stroke</code> (リストボックス)	オブジェクト内で使用されるフォントや線のカラーを指定します。	任意のcss値; "transparent"; "automatic"
<code>strokeDashArray</code>	点線のタイプを、点と白のパターンにより指定します。	数値配列または文字列
<code>strokeWidth</code>	線の幅を指定します。	整数、または0(印刷されるフォームにおける最小幅)
<code>style</code>	ボタンの外観を設定します。詳細についてはボタンスタイルを参照ください。	"regular", "flat", "toolbar", "bevel", "roundedBevel", "gradientBevel", "texturedBevel", "office", "help", "circular", "disclosure", "roundedDisclosure", "custom"
<code>styledText</code>	選択エリアでスタイルの利用を可能にするかどうかを指定します。	true, false
<code>switchBackWhenReleased</code>	ユーザーがボタンをクリックしているとき以外は、一番目のピクチャーが常に表示されます。ボタンがクリックされると、マウスボタンが放されるまで二番目のピクチャーが表示されます。	true, false
<code>switchContinuously</code>	ユーザーがマウスボタンを押している間は、各ピクチャーが連続的に(アニメーションのように)表示されます。	true, false
<code>switchWhenRollover</code>	マウスカーソルが通過すると、ピクチャーボタンの内容が変わります。カーソルがボタンエリアを離れるとき、最初のピクチャーが再度表示されます。	true, false
<code>t</code>		
<code>table</code>	リストサブフォームが属するテーブル(あれば)	4D テーブル名、または""
<code>text</code>	フォームオブジェクトのタイトル	なんらかのテキスト
<code>textAlign</code>	エリア中のテキストの横位置を指定します	"automatic", "right", "center", "justify", "left"

プロパティ	説明	とりうる値
<code>textAngle</code>	テキストエリアの角度 (回転) を変更します。	0, 90, 180, 270
<code>textDecoration</code>	テキストの下に線を引きます。	"normal", "underline"
<code>textFormat</code>	表示や印刷時に文字フィールドや変数にデータを表示する方法を制御します。	"#### ####", "(###) ### ####", "### #### ####", "# ## ## ## ##", "00000", カスタムフォーマット
<code>textPlacement</code>	アイコンに対するボタンタイトルの相対的な位置を指定します。	"left", "top", "right", "bottom", "center"
<code>threeState</code>	チェックボックスオブジェクトに、3 番目の状態を付加します。	true, false
<code>timeFormat</code>	表示/印刷時に時間をどのように表示かを管理します。4D にビルトインされているフォーマットから選択されなければなりません。	"systemShort", "systemMedium", "systemLong", "iso8601", "hh_mm_ss", "hh_mm", "hh_mm_am", "mm_ss", "HH_MM_SS", "HH_MM", "MM_SS", "blankIfNull" (他の値と組み合わせることができます)
<code>truncateMode</code>	リストボックスのカラムが、中身をすべて表示するのには狭すぎる場合の値の表示を管理します。	"withEllipsis", "none"
<code>type</code>	必須設定です。フォームオブジェクトのタイプを指定します。	"text", "rectangle", "groupBox", "tab", "line", "button", "checkbox", "radio", "dropdown", "combo", "webArea", "write", "subform", "plugin", "splitter", "buttonGrid", "progress", "ruler", "spinner", "stepper", "list", "pictureButton", "picturePopup", "listbox", "input", "view"
<code>tooltip</code>	ユーザーに対して、フィールドについての追加情報を提供します。	ユーザー用のヘルプ情報のテキスト
<code>top</code>	フォーム上のオブジェクトの上の座標	最小値: 0
u		
<code>urlSource</code>	Webエリアにロードされた、またはロード中の URL	URL
<code>useLastFrameAsDisabled</code>	ボタンが無効な場合に表示するサムネールとして、最後のサムネールを使用します。	true, false
<code>userInterface</code>	4D View Pro エリアに使用するインターフェース	"none" (デフォルト), "ribbon", "toolbar"
v		
<code>values</code>	リストボックス列にしようするデフォルト値のリスト	例: "A", "B", "42"...
<code>variableCalculation</code>	数値の計算を実行することができます。	"none", "minimum", "maximum", "sum", "count", "average", "standardDeviation", "variance", "sumSquare"
<code>verticalAlign</code>	エリア中のテキストの縦位置を指定します。	"automatic", "top", "middle", "bottom"
<code>verticalLineStroke</code>	リストボックス内の縦線の色を指定します (デフォルトはグレー)。	任意の css 値; "transparent"; "automatic"
<code>visibility</code>	アプリケーションモードでオブジェクトが非表示になります。	"visible", "hidden", "selectedRows", "unselectedRows"
w		

<code>webEngine</code>	説明 Webエリアで使用する描画エンジンを 2つのうちから選択します。	可能値 "embedded", "system"
<code>width</code>	オブジェクトの横のサイズを指定します。	最小値: 0
<code>withFormulaBar</code>	4D View Pro エリアにおいて、ツールバーのすぐ下にフォーミュラバーを表示します。	true, false
<code>wordwrap</code>	このオプションは、表示する内容がオブジェクトの幅を超えたときの表示を管理します。	"automatic" (リストボックスを除く), "normal", "none"
<code>z</code>		
<code>zoom</code>	4D Write Pro エリアのズーム率を設定します。	数値 (最小値 = 0)

アクション

ドラッグ有効

ユーザーによるオブジェクトのドラッグを制御します。デフォルトでは、ドラッグ操作は禁止されています。

二つのドラッグモードが提供されています：

- カスタム：このモードでは、オブジェクトに対しておこなわれたドラッグ操作は、当該オブジェクトのコンテキストにおいて `On Begin Drag` フォームイベントを発生させます。これを利用して、開発者はメソッドを用いてドラッグアクションを管理しなければなりません。
つまり、カスタムモードにおいては、ドラッグ & ドロップ操作のすべてが開発者により管理されます。このモードでは、ドラッグ & ドロップに基づいたあらゆるインターフェースを実装することができます。これにはデータの転送を必ずしも伴わないものも含まれ、ファイルを開くや計算をトリガーするなどの任意のアクションを実行することができます。このモードは専用のプロパティ、イベント、ペーストボード テーマのコマンド等の組み合わせに基づいています。
- 自動：このモードでは、ドラッグ元のフォームオブジェクトからテキストやピクチャーが 4D によって コピー されます。このコピーは、同じ 4Dエリア内、2つの 4Dエリア間、4D と他のアプリケーション間で使用できます。たとえば、自動ドラッグ (& ドロップ) を使用して、プログラムを使用せず、2つのフィールド間で値をコピーできます：



このモードでは、`On Begin Drag` フォームイベントは生成されません。自動ドラッグが有効のときに標準のドラッグを "強制" したい場合、アクションの間 `Alt` (Windows) または `Option` (macOS) キーを押しながら操作します。このオプションはピクチャーでは利用できません。

詳細については 4D ランゲージリファレンス マニュアルの [ドラッグ & ドロップ](#) を参照してください。

JSON 文法

名称	データタイプ	とりうる値
dragging	テキスト	"none" (デフォルト), "custom", "automatic" (リストボックスを除く)

対象オブジェクト

[4D Write Pro エリア](#) - [入力](#) - [階層リスト](#) - [リストボックス](#) - [プラグインエリア](#)

参照

ドロップ有効

ユーザーがドラッグ & ドロップしたデータをオブジェクトが受け取ることができるかどうかを制御します。

二つのドロップモードが提供されています：

- カスタム：このモードでは、オブジェクトに対しておこなわれたドロップ操作は、当該オブジェクトのコンテキストにおいて `On Drag Over` と `On Drop` フォームイベントを発生させます。これを利用して、開発者はメソッドを用いてドロップアクションを管理しなければなりません。
つまり、カスタムモードにおいては、ドラッグ & ドロップ操作のすべてが開発者により管理されます。このモードでは、ドラッグ & ドロップに基づいたあらゆるインターフェースを実装することができます。これにはデータの転送を必ずしも伴わないものも含まれ、ファイルを開くや計算をトリガーするなどの任意のアクションを実行することができます。このモードは専用のプロパティ、イベント、ペーストボード テーマのコマンド等の組み合わせに基づいています。
- 自動：このモードでは、4D は可能な限り自動で、オブジェクトにドロップされたテキストやピクチャー型データの挿入を管理します (データはオブジェクトにペーストされます)。このモードでは、`On Drag Over` と `On Drop` フォームイベントは生成されません。他方、ドロップ中の `On After Edit` とオブジェクトがフォーカスを失った時の `On Data Change` イベントは生成されます。

詳細については 4D ランゲージリファレンス マニュアルの [ドラッグ & ドロップ](#) を参照してください。

JSON 文法

名称	データタイプ	とりうる値
dropping	テキスト	"none" (デフォルト), "custom", "automatic" (リストボックスを除く)

対象オブジェクト

[4D Write Pro エリア](#) - [ボタン](#) - [入力](#) - [階層リスト](#) - [リストボックス](#) - [プラグインエリア](#)

参照

[ドラッグ有効](#)

オブジェクトメソッド実行

このオプションを選択した場合、ユーザーがインジケーターの値を変更すると同時に `On Data Change` イベントが生成され、オブジェクトメソッドが実行されます。デフォルトでは、変更後にメソッドが実行されます。

JSON 文法

名称	データタイプ	とりうる値
continuousExecution	boolean	true, false

対象オブジェクト

[進捗インジケーター](#) - [ルーラー](#) - [ステッパー](#)

メソッド

オブジェクトに関連づけられたメソッドへの参照。オブジェクトメソッドは通常、フォームが表示または印刷されている間、オブジェクトを "管理" します。オブジェクトメソッドは呼び出す必要がありません。オブジェクトメソッドが関連づけられているオブジェクトに関わるイベントが発生した場合、4D は自動的にオブジェクトメソッドを呼び出します。

メソッド参照にはいくつかのタイプが利用可能です:

- 標準のオブジェクトメソッドファイルパス:
`ObjectMethods/objectName.4dm` (`objectName` には実際の [オブジェクト名](#) が入ります)。このタイプの参照は、当該メソッドファイルがデフォルトの場所 (`"sources/forms/formName/ObjectMethods/"`) にあることを示します。この場合、エディター上でフォームオブジェクトに対して操作（名称変更、複製、コピー/ペーストなど）がおこなわれると、4D はこれらの変更を自動的にオブジェクトメソッドに反映させます。
- 拡張子を省いた既存のプロジェクトメソッド名: `myMethod`。この場合、フォームオブジェクトに対して操作がおこなわれても、4D はそれらの変更を自動反映しません。
- .4dm 拡張子を含むカスタムのメソッドファイルパス:
`.../../CustomMethods/myMethod.4dm`。ファイルシステムも使用できます:
`/RESOURCES/Buttons/bOK.4dm`。この場合、フォームオブジェクトに対して操作がおこなわれても、4D はそれらの変更を自動反映しません。

JSON 文法

名称	データタイプ	とりうる値
method	テキスト	オブジェクトメソッドの標準またはカスタムのファイルパス、またはプロジェクトメソッド名

対象オブジェクト

4D View Pro エリア - 4D Write Pro エリア - ボタン - ボタングリッド - チェックボックス - コンボボックス - ドロップダウンリスト - フォーム - 階層リスト - 入力 - リストボックス - リストボックス列 - ピクチャーボタン - ピクチャーポップアップメニュー - ブラグインエリア - 進捗インジケーター - ラジオボタン - ルーラー - スピナー - スプリッター - ステッパー - サブフォーム - タブコントロール - Web エリア

行の移動可

配列型リストボックス

ランタイムにおける行の移動を許可します。このオプションはデフォルトで選択されています。 [セレクション型のリストボックス](#) および [階層リストボックス](#) では、このオプションは提供されていません。

JSON 文法

名称	データタイプ	とりうる値
movableRows	boolean	true, false

対象オブジェクト

リストボックス

複数選択可

[階層リスト](#)において、複数項目の選択を許可します。

JSON 文法

名称	データタイプ	とりうる値
selectionMode	テキスト	"multiple", "single", "none"

対象オブジェクト

階層リスト

ソート可

[リストボックス](#) ヘッダーのクリックによる列データの並び替えを有効にします。このオプションはデフォルトで選択されています。ピクチャー型配列（列）はこのメカニズムではソートできません。

レコードセレクションに基づリストボックスの場合、標準のソート機能は以下の場合のみ有効です：

- データソースが カレントセレクション であり、
- その列にフィールドが割り当てられていること（文字、数値、日付、時間、およびブール型）。

他の場合（命名セレクションに基づリストボックスや、式が割り当てられた列）、標準のソート機能は動作しません。標準のリストボックスソートは、データベースのカレントセレクションの順番を変更します。しかし、ハイライトされたレコードとカレントレコードは変更されません。標準の並び替えは、リストボックスのすべての列（式が割り当てられた列も含む）を同期します。

JSON 文法

名称	データタイプ	とりうる値
sortable	boolean	true, false

対象オブジェクト

リストボックス

標準アクション

アクティブオブジェクトにより実行される典型的な処理（例：レコードの入力・取り消し・削除、レコード間の移動、マルチページフォームでのページ間の移動、など）は、4D より標準アクションとして提供されています。詳細な情報に関しては、デザインリファレンスの [標準アクション](#) の章を参照ください。

フォームオブジェクトには、標準アクションとメソッドの両方を割り当てるすることができます。この場合、標準アクションは通常、メソッドの後に実行されます。また、4D はこのアクションを使用して、カレントコンテキストに応じてオブジェクトを有効化/無効化します フォームオブジェクトが無効化されていると、関連づけられたメソッドは実行されません。

このプロパティは `OBJECT SET ACTION` コマンドによって設定することができます。

JSON 文法

名称	データタイプ	とりうる値
action	string	有効な 標準アクション

対象オブジェクト

[ボタン](#) - [ボタングリッド](#) - [チェックボックス](#) - [ドロップダウンリスト](#) - [リストボックス](#) - [ピクチャーボタン](#) - [ピクチャーポップアップメニュー](#) - [タブコントロール](#)

アニメーション

先頭フレームに戻る

ピクチャーを連続的に表示し続けます。最後のピクチャーに到達して再度クリックすると、最初のピクチャーが表示されます。

JSON 文法

名称	データタイプ	とりうる値
loopBackToFirstFrame	boolean	true, false

対象オブジェクト

[ピクチャーボタン](#)

マウスアップで戻る

ユーザーがボタンをクリックしているとき以外は、一番目のピクチャーが常に表示されます。ボタンがクリックされると、マウスボタンが放されるまで二番目のピクチャーが表示されます。このモードを使用すると、それぞれの状態（アイドルとクリック）ごとに異なるピクチャーを使用した動作ボタンを作成することができます。このモードを使って 3D 効果を作成したり、ボタンの動作を表現するピクチャーを表示することができます。

JSON 文法

名称	データタイプ	とりうる値
switchBackWhenReleased	boolean	true, false

対象オブジェクト

[ピクチャーボタン](#)

マウス押下中は自動更新

ユーザーがマウスボタンを押している間は、各ピクチャーが連続的に（アニメーションのように）表示されます。最後のピクチャーに達しても、オブジェクトは最初のピクチャーに戻りません。

JSON 文法

名称	データタイプ	とりうる値
switchContinuously	boolean	true, false

対象オブジェクト

[ピクチャーボタン](#)

アニメーション間隔 (tick)

このモードを使用すると、一定のスピードで (tick 単位) ピクチャーボタンの内容が繰り返し表示されます。このモードでは、他のすべてのオプションが無視されます。

JSON 文法

名称	データタイプ	とりうる値
frameDelay	integer	最小値: 0

対象オブジェクト

[ピクチャーボタン](#)

ロールオーバー効果

マウスカーソルが通過すると、ピクチャーボタンの内容が変わります。カーソルがボタンエリアを離れるとき、最初のピクチャーが再度表示されます。

JSON 文法

名称	データタイプ	とりうる値
switchWhenRollover	boolean	true, false

対象オブジェクト

[ピクチャーボタン](#)

無効時に最終フレームを使用

ボタンが無効な場合に表示するサムネールとして、最後のサムネールを使用します。ボタンが使用不可の場合に用いられるサムネールは、4Dにより別に処理されます。このオプションと “マウス押下中は自動更新” および “先頭フレームへ戻る” オプションを組み合わせると、最終ピクチャーはボタンに割り当てられた順序から外され、無効時にのみ表示されるようになります。

JSON 文法

名称	データタイプ	とりうる値
useLastFrameAsDisabled	boolean	true, false

対象オブジェクト

[ピクチャーボタン](#)

アピアランス

デフォルトボタン

フォーム上のボタンのいずれにも [フォーカス可](#) プロパティが設定されていない場合、デフォルトボタンプロパティが有効化されたボタンがランタイムにおいて最初のフォーカスを得ます。

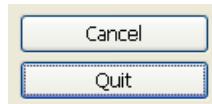
デフォルトボタンは、フォームページごとに一つのみ設定できます。

macOS上では、デフォルトボタンプロパティによってボタンの見た目が変更され、推奨されている選択肢であることをユーザーに対して示します。デフォルトボタンとフォーカスされているボタンは同一である必要はありません。macOSにおけるデフォルトボタンは特徴的な青い外観を持ちます：



このデフォルトボタンとしての外観を得るには、ボタンの高さが 22以下に設定されている必要があります。

Windows上では "推奨選択肢" の概念はサポートされていないため、ランタイムにおいて異なる外観を持つのはフォーカスされているボタンのみですが、4D フォームエディター上ではデフォルトボタンは青い枠線で表されます：



JSON 文法

名称	データタイプ	とりうる値
defaultButton	boolean	true, false

対象オブジェクト

[通常ボタン](#) - [フラットボタン](#)

フォーカスの四角を隠す

ランタイムにおいて、タブキーやシングルクリックによってフォーカスを得たフィールドや入力可能なエリアは、選択状態を示す四角で縁取りされます。このプロパティを使用して、フォーカスの四角を非表示にできます。特定のインターフェースにおいては、フォーカスの四角を非表示にすることが便利かもしれません。

JSON 文法

名称	データタイプ	とりうる値
hideFocusRing	boolean	true, false

対象オブジェクト

[4D Write Pro エリア](#) - [階層リスト](#) - [入力](#) - [リストボックス](#) - [サブフォーム](#)

セレクションハイライトを非表示

セレクション型リストボックス

リストボックスのセレクションハイライトを非表示にします。

このオプションが有効化されると、リストボックスにおける行選択を可視化するセレクションハイライトが非表示になります。ハイライトが非表示になっていても選択行は引き続き機能的に有効です。しかしながら、画面上では選択状態が明示されなくなるため、[プログラムによって選択行を可視化](#)する必要があります。

デフォルトでは、このオプションは有効化されていません。

JSON 文法

名称	データタイプ	とりうる値
hideSystemHighlight	boolean	true, false

対象オブジェクト

リストボックス

横スクロールバー

表示エリアを左右に移動できるようにするインターフェースツールです。

使用可能な値:

プロパティ リスト	JSON 値	説明
○	"visible"	スクロールバーは必要のない場合でも常に表示されます。つまり、オブジェクトのコンテンツのサイズがフレームのサイズより小さい場合でも表示されます。
×	"hidden"	スクロールバーは表示されません。
自動	"automatic"	スクロールバーは必要なときに表示されます。つまり、オブジェクトのコンテンツのサイズがフレームのサイズより大きい場合には表示されます。

ピクチャ型のオブジェクトは、表示フォーマットが "トランケート (中央合わせなし)" に設定されているときに、スクロールバーを持つことができます。

JSON 文法

名称	データタイプ	とりうる値
scrollbarHorizontal	テキスト	"visible", "hidden", "automatic"

対象オブジェクト

階層リスト - サブフォーム - リストボックス - 入力 - 4D Write Pro エリア

参照

縦スクロールバー

解像度

4D Write Pro エリアの画面解像度を設定します。デフォルト値は 72dpi (macOS) で、これはすべてのプラットフォームにおける 4D フォームの標準解像度です。96dpi に指定すると、Windows/Web レンダリングを macOS および Windows の両プラットフォームに設定します。この項目を自動に設定すると、macOS と Windows 間でドキュメントのレンダリングが異なることになります。

JSON 文法

名称	データタイプ	とりうる値
dpi	number	0=automatic, 72, 96

対象オブジェクト

4D Write Pro エリア

背景を表示

ページの背景画像および背景色を表示/非表示にします。

JSON 文法

名称	データタイプ	とりうる値
showBackground	boolean	true (デフォルト), false

対象オブジェクト

4D Write Pro エリア

フッター表示

ビューモード が "ページ" に設定されている場合に、ページのフッターを表示/非表示にします。

JSON 文法

名称	データタイプ	とりうる値
showFooters	boolean	true (デフォルト), false

対象オブジェクト

4D Write Pro エリア

フォーミュラバーを表示

有効化すると、4D View Pro エリアにおいてツールバーのすぐ下にフォーミュラバーが表示されます。選択されていない場合、フォーミュラバーは非表示となります。

このプロパティは [ツールバー](#) インターフェースの場合に利用可能です。

JSON 文法

名称	データタイプ	とりうる値
withFormulaBar	boolean	true (デフォルト), false

対象オブジェクト

ヘッダーを表示

ビューモードが "ページ" に設定されている場合に、ページのヘッダーを表示/非表示にします。

JSON 文法

名称	データタイプ	とりうる値
showHeaders	boolean	true (デフォルト), false

対象オブジェクト

[4D Write Pro エリア](#)

非表示文字を表示

非表示の文字を表示/非表示にします。

JSON 文法

名称	データタイプ	とりうる値
showHiddenChars	boolean	true (デフォルト), false

対象オブジェクト

[4D Write Pro エリア](#)

水平ルーラーを表示

ドキュメントビューガ "ページモード" の場合に、水平ルーラーを表示/非表示にします。

JSON 文法

名称	データタイプ	とりうる値
showHorizontalRuler	boolean	true (デフォルト), false

対象オブジェクト

[4D Write Pro エリア](#)

HTML WYSIWIG 表示

HTML WYSIWYG ビューを有効/無効にします。このビューでは、すべてのブラウザーに対応していない 4D Write Pro の属性が取り除かれます。

JSON 文法

名称	データタイプ	とりうる値
showHTMLWysiwyg	boolean	true, false (デフォルト)

対象オブジェクト

[4D Write Pro エリア](#)

ページフレームを表示

[ビューモード](#) が "ページ" に設定されている場合に、ページのフレームを表示/非表示にします。

JSON 文法

名称	データタイプ	とりうる値
showPageFrames	boolean	true, false

対象オブジェクト

[4D Write Pro エリア](#)

参照を表示

ドキュメントに 参照 として挿入された 4D式をすべて表示します。このオプションが無効になっていると、4D Write Pro は挿入された 4D式をカレント値で評価して、その値 を表示します。4Dフィールドまたは式を挿入すると、4D Write Pro はデフォルトでそのカレント値を表示します。もとのフィールドや式を確認したいときには、このオプションを有効にします。すると、これらの参照は灰色の背景色とともにドキュメント内に表示されます。

たとえば、フォーマットを指定したカレント日付を挿入していると、デフォルトでは次の表示になります：

July 11, 2016

参照を表示オプションを有効にすると、代わりにもとの参照が表示されます：

String(Current date;Internal date long)

4D 式を挿入するには、`ST_INSERT_EXPRESSION` コマンドを使います。

JSON 文法

名称	データタイプ	とりうる値
showReferences	boolean	true, false (デフォルト)

対象オブジェクト

[4D Write Pro エリア](#)

垂直ルーラーを表示

ドキュメントビューが [ページモード](#) の場合に、垂直ルーラーを表示/非表示にします。

JSON 文法

名称	データタイプ	とりうる値
showVerticalRuler	boolean	true (デフォルト), false

対象オブジェクト

[4D Write Pro エリア](#)

タブコントロールの位置

フォーム上のタブコントロールの位置を指定することができます。このプロパティにはすべてのプラットフォームからアクセスできますが、macOS 上でのみ動作します。タブコントロールは上（標準）、または下に配置することができます。

位置がカスタマイズされたタブコントロールを Windows で表示すると、自動的に標準の位置（上）に戻されます。

JSON 文法

名称	データタイプ	とりうる値
labelsPlacement	boolean	"top", "bottom"

対象オブジェクト

[タブコントロール](#)

ユーザーインターフェース

4D View Pro エリアにインターフェースを追加することで、エンドユーザーが基本的な編集とデータ操作をおこなえるようになります。4D では 2種類のインターフェース（リボンと ツールバー）を提供しており、そのどちらかを選ぶことができます。

JSON 文法

名称	データタイプ	とりうる値
userInterface	text	"none" (デフォルト), "ribbon", "toolbar"

対象オブジェクト

[4D View Pro エリア](#)

参照

[4D View Pro リファレンスガイド](#)

縦スクロールバー

表示エリアを上下に移動できるようにするインターフェースツールです。

使用可能な値:

プロパティリスト	JSON 値	説明
○	"visible"	スクロールバーは必要のない場合でも常に表示されます。つまり、オブジェクトのコンテンツのサイズがフレームのサイズより小さい場合でも表示されます。
×	"hidden"	スクロールバーは表示されません。
自動	"automatic"	スクロールバーは必要なときに表示されます。つまり、オブジェクトのコンテンツのサイズがフレームのサイズより大きい場合には表示されます。

ピクチャー型のオブジェクトは、表示フォーマットが "トランケート (中央合わせなし)" に設定されているときに、スクロールバーを持つことができます。

テキスト入力オブジェクトにスクロールバーがない場合、矢印キーを使用してスクロールできます。

JSON 文法

名称	データタイプ	とりうる値
scrollbarVertical	テキスト	"visible", "hidden", "automatic"

対象オブジェクト

[階層リスト](#) - [サブフォーム](#) - [リストボックス](#) - [入力](#) - [4D Write Pro エリア](#)

参照

[横スクロールバー](#)

ビューモード

フォームエリア内の 4D Write Pro ドキュメントの表示モードを設定します。次の値が提供されています:

- ページ: もっとも完全といえるビューモードで、ページの枠、余白、改ページ、ヘッダー & フッターなどを含みます。
- 下書き: 基本のドキュメントプロパティを含む下書きモードです。
- 埋め込み: 埋め込みエリアに適切なビューモードです。余白や、ヘッダー & フッター、ページフレームなどは表示されません。このモードは Web に似た出力をするのに使用することもできます (この場合には、[解像度を 96dpi](#) に設定のうえ、[HTML WYSIWYG 表示](#) オプションを有効にします)。

ビューモードプロパティは画面上のレンダリングにのみ使用されます。印刷設定については、専用のレンダリングルールが自動的に適用されます。

JSON 文法

名称	データタイプ	とりうる値
layoutMode	text	"page", "draft", "embedded"

対象オブジェクト

[4D Write Pro エリア](#)

拡大

4D Write Pro エリアのコンテンツ表示に使用するズーム率を設定します。

JSON 文法

名称	データタイプ	とりうる値
zoom	number	minimum = 0

対象オブジェクト

[4D Write Pro エリア](#)

背景色と境界線

交互に使用する背景色

奇数番の行/列に使用するための異なる背景色を設定することができます。デフォルトでは、自動 が選択されており、リストボックスレベルで設定されている "交互に使用する背景色" を列も使用します。

JSON 文法

名称	データタイプ	とりうる値
alternateFill	string	任意の CSS値; "transparent"; "automatic"; "automaticAlternate"

対象オブジェクト

[リストボックス - リストボックス列](#)

背景色/塗りカラー

オブジェクトの背景色を設定します。

リストボックスの場合にはデフォルトで、自動 が選択されており、リストボックスレベルで設定されている背景色を列も使用します。

JSON 文法

名称	データタイプ	とりうる値
fill	string	任意の css値; "transparent"; "automatic"

対象オブジェクト

[階層リスト - リストボックス - リストボックス列 - リストボックスフッター - 楠円 - 四角 - テキストエリア](#)

参照

[透過](#)

背景色式

セレクションとコレクション型リストボックス

リストボックスの各行にカスタムの背景色を指定するための式または変数 (配列変数は使用不可)。式または変数は表示行ごとに評価され、RGB値を返さなくてはなりません。詳細については、4Dランゲージリファレンス マニュアルの [OBJECT SET RGB COLORS](#) コマンドの説明を参照ください。

また、このプロパティは `LISTBOX SET PROPERTY` コマンドに `lk background color expression` 定数を指定して設定することもできます。

コレクション/エンティティセレクション型リストボックスでは、このプロパティは [メタ情報式](#) を使用しても設定することができます。

JSON 文法

名称	データタイプ	とりうる値
rowFillSource	string	RGBカラー値を返す式

対象オブジェクト

[リストボックス - リストボックス列](#)

境界線スタイル

リストボックスの境界線のスタイルを設定します。

JSON 文法

名称	データタイプ	とりうる値
borderStyle	テキスト	"system", "none", "solid", "dotted", "raised", "sunken", "double"

対象オブジェクト

[4D View Pro エリア](#) - [4D Write Pro エリア](#) - [ボタン](#) - [ボタングリッド](#) - [階層リスト](#) - [入力](#) - [リストボックス](#) - [ピクチャーボタン](#) - [ピクチャーポップアップメニュー](#) - [プラグインエリア](#) - [進捗インジケーター](#) - [ルーラー](#) - [スピナー](#) - [ステッパー](#) - [サブフォーム](#) - [テキストエリア](#) - [Web エリア](#)

点線タイプ

点線のタイプを、点と白のパターンにより指定します。

JSON 文法

名称	データタイプ	とりうる値
strokeDashArray	数値配列または文字列	例: 6個の点と1個の空白のパターンは "6 1" または [6,1] によって表します。

対象オブジェクト

[四角](#) - [橢円](#) - [線](#)

追加の空白の行を非表示

リストボックスオブジェクト下部に追加される余分な空白行の表示を管理します。デフォルトで、4D は空のエリアを埋めるためにこのような行を追加します:

Days	Values
Monday	5
Tuesday	6
Wednesday	41
Thursday	66
Friday	12
Saturday	2
Sunday	88

Extra blank rows

このオプションをチェックすると、これらの空白行を除去することができます。リストボックスオブジェクトの下部は空のままになります:

Days	Values
Monday	5
Tuesday	6
Wednesday	41
Thursday	66
Friday	12
Saturday	2
Sunday	88

JSON 文法

名称	データタイプ	とりうる値
hideExtraBlankRows	boolean	true, false

対象オブジェクト

[リストボックス](#)

線カラー

オブジェクトの線の色を指定します。カラーは次の方法で指定できます:

- カラーネーム - 例: "red"
- 16進数値 - 例: "#ff0000"
- RGB値 - 例: "rgb(255,0,0)"

このプロパティは OBJECT SET RGB COLORS** コマンドによって設定することができます。

JSON 文法

名称	データタイプ	とりうる値
stroke	string	任意の css値; "transparent"; "automatic"

このプロパティはテキスト系のオブジェクトでも利用可能です。この場合、このプロパティはフォントカラーおよびオブジェクトの線カラーの両方を指定します ([フォントカラー 参照](#))。

対象オブジェクト

[線 - 橋円 - 四角](#)

線幅

線の幅を指定します。

JSON 文法

名称	データタイプ	とりうる値
strokeWidth	number	印刷されるフォームにおける最小幅 0 から、整数値 < 20 まで

対象オブジェクト

線 - 楕円 - 四角

行背景色配列

配列型リストボックス

リストボックスまたはリストボックス列の各行にカスタムの背景色を適用するのに使用する配列名です。

倍長整数型の配列の名前を入力しなければなりません。配列のそれぞれの要素はリストボックスの行（あるいは列のセル）に対応します。つまりこの配列は、各列に関連づけられている配列と同じサイズでなければいけません。ここでは [SET RGB COLORS](#) テーマの定数を使用することができます。もし上レベルで定義されている背景色をそのままセルに継承したい場合には、対応する配列の要素に -255 を渡します。

たとえば、リストボックスプロパティにてグレー/ライトグレーカラーが行の交互背景色として設定されているとします。同じリストボックスに行背景色配列が指定されており、行内で負の値が一つでもあれば色をオレンジに変えます：

```
<>_BgnColors{$i}:=0x00FFD0B0 // オレンジ
<>_BgnColors{$i}:=-255 // デフォルト値
```

Header1	Header2	Header3
21483	9031	27290
24151	21990	-923
21351	2982	18009
8089	12898	20941
13001	-802	22059
4321	16826	11303
24082	26214	22380
16680	23651	20403
-2678	24818	29896
25639	2691	9687
28794	26941	21486
26083	21092	13476
27928	4092	15441
19987	28211	21191
7996	6300	4089
-1063	13388	23683
13008	7470	19897
5388	26918	13547
28559	27007	8365
28454	22646	13824

次に、負の値を持つセルの色を濃いオレンジで示したい場合、各列にも行背景色配列を設定します（例：<>_BgnColor_1, <>_BgnColor_2 と <>_BgnColor_3）。これらの配列の値は、リストボックスプロパティに設定されているものや、全体用の行背景色配列よりも優先されます。

```
<>_BgnColorsCol_3{2}:=0x00FF8000 // 濃いオレンジ
<>_BgnColorsCol_2{5}:=0x00FF8000
<>_BgnColorsCol_1{9}:=0x00FF8000
<>_BgnColorsCol_1{16}:=0x00FF8000
```

Header1	Header2	Header3
21483	9031	27290
24151	21990	-923
21351	2982	18009
8089	12898	20941
13001	-802	22059
4321	16826	11303
24082	26214	22380
16680	23651	20403
-2678	24818	29896
25639	2691	9687
28794	26941	21486
26083	21092	13476
27928	4092	15441
19987	28211	21191
7996	6300	4089
-1063	13388	23683
13008	7470	19897
5388	26918	13547
28559	27007	8365
28454	22646	13824

`LISTBOX SET ROW FONT STYLE` や `LISTBOX SET ROW COLOR` コマンドを使っても同じような効果が得られます。コマンドを使う利点は、スタイル/カラー配列をあらかじめ列に設定する必要がないことです。この場合、これらはコマンドによって動的に作成されます。

JSON 文法

名称	データタイプ	とりうる値
rowFillSource	string	倍長整数型配列の名前

対象オブジェクト

[リストボックス - リストボックス列](#)

透過

リストボックスの背景を透明にします。このプロパティが有効になっていると、列に対して設定されている [交互に使用する背景色](#) および [背景色](#) の設定は無視されます。

JSON 文法

名称	データタイプ	とりうる値
fill	テキスト	"transparent"

対象オブジェクト

[リストボックス](#)

参照

[背景色/塗りカラー](#)

座標とサイズ

自動行高

このプロパティは、配列型かつ階層のないリストボックスにおいてのみ使用可能です。このプロパティはデフォルトではチェックされていません。

このプロパティが有効化されると、カラムの内容に応じて各行の高さが 4D によって自動的に計算されます 行の高さを計算する際には、このオプションがチェックされているカラムのみが考慮されることに注意が必要です。

リストボックスの [横方向サイズ変更](#) プロパティに "拡大" を設定している場合にフォームをリサイズすると、一番右のカラムの幅は必要に応じて最大幅を超えて拡大されます。

このプロパティが有効化されると、セルの内容がすべて表示され、切り落とされることがないように各行の高さが自動的に計算されます (ただし [ワードラップ](#) オプションが無効化されている場合を除きます)。

- 行の高さを計算する際には、以下のものが考慮されます:
 - 中身の型 (テキスト、数値、日付、時間、ピクチャー (計算結果はピクチャーフォーマットによります)、オブジェクト)
 - コントロールの型 (入力、チェックボックス、リスト、ドロップダウン)
 - フォント、フォントスタイル、フォントサイズ
 - [ワードラップ](#) オプション: 無効化されている場合、高さは段落の数に応じます (行は切り落とされます)。有効化されている場合、高さは行数に応じます (切り落とされません)。
- 行の高さを計算する際には、以下のものは考慮されません:
 - 非表示のカラムの中身
 - プロパティリスト内、あるいはプログラミングによって設定された [行の高さ](#) および [行高さ配列](#) プロパティ (あつた場合)

自動行高オプションを有効化すると、ランタイムにおいて追加の計算が必要となるため、とくにリストボックスが大量の行数を持つ場合に、スクロール時のスムーズさに影響が出る可能性があります。

JSON 文法

名称	データタイプ	とりうる値
rowHeightAuto	boolean	true, false

対象オブジェクト

リストボックス列

下

フォーム上のオブジェクトの下の座標。

JSON 文法

名称	データタイプ	とりうる値
bottom	number	最小値: 0

対象オブジェクト

4D View Pro エリア - 4D Write Pro エリア - ボタン - ボタングリッド - チェックボックス - コンボボックス - ドロップダウンリスト - グループボックス - 階層リスト - 入力 - リストボックス - 線 - リストボックス列 - 楕円 - ピクチャーボタン - ピクチャーポップアップメニュー - ブラグインエリア - 進捗インジケーター - ラジオボタン - 四角 - ルーラー - スピナー - スプリッター - スタティックピクチャー - ステッパー - サブフォーム - タブコントロール - テキストエリア - Web エリア

左

フォーム上のオブジェクトの左の座標。

JSON 文法

名称	データタイプ	とりうる値
left	number	最小値: 0

対象オブジェクト

4D View Pro エリア - 4D Write Pro エリア - ボタン - ボタングリッド - チェックボックス - コンボボックス - ドロップダウンリスト - グループボックス - 階層リスト - 入力 - リストボックス - 線 - リストボックス列 - 楕円 - ピクチャーボタン - ピクチャーポップアップメニュー - ブラグインエリア - 進捗インジケーター - ラジオボタン - ルーラー - 四角 - スピナー - スプリッター - スタティックピクチャー - ステッパー - サブフォーム - タブコントロール - テキストエリア - Web エリア

右

フォーム上のオブジェクトの右の座標。

JSON 文法

名称	データタイプ	とりうる値
right	number	最小値: 0

対象オブジェクト

4D View Pro エリア - 4D Write Pro エリア - ボタン - ボタングリッド - チェックボックス - コンボボックス - ドロップダウンリスト - グループボックス - 階層リスト - 入力 - リストボックス - 線 - リストボックス列 - 楕円 - ピクチャーボタン - ピクチャーポップアップメニュー - ブラグインエリア - 進捗インジケーター - ラジオボタン - ルーラー - 四角 - スピナー - スプリッター - スタティックピクチャー - ステッパー - サブフォーム - タブコントロール - テキストエリア - Web エリア

上

フォーム上のオブジェクトの上の座標。

JSON 文法

名称	データタイプ	とりうる値
top	number	最小値: 0

対象オブジェクト

4D View Pro エリア - 4D Write Pro エリア - ボタン - ボタングリッド - チェックボックス - コンボボックス - ドロップダウンリスト - グループボックス - 階層リスト - 入力 - リストボックス - 線 - リストボックス列 - 楕円 - ピクチャーボタン - ピクチャーポップアップメニュー - ブラグインエリア - 進捗インジケーター - ラジオボタン - ルーラー - 四角 - スピナー - スプリッター - スタティックピクチャー - ステッパー - サブフォーム - タブコントロール - テキストエリア - Web エリア

角の半径

四角 型の図形について、角の丸みをピクセル単位で指定します。デフォルトでは、四角の角の半径は 0ピクセルとなっています。このプロパティを変更することによって独自の形の角の丸い四角を描画することができます:



最小値は 0 で、この場合には標準の（角の丸くない）四角が描画されます。最大値は四角のサイズに応じて変化し、動的に計算されます（ただし四角の短辺の半分を超えることはできません）。

このプロパティは、OBJECT Get corner radius と OBJECT SET CORNER RADIUS コマンドを使用して設定することもできます。

JSON 文法

名称	データタイプ	とりうる値
borderRadius	integer	最小値: 0

対象オブジェクト

四角

高さ

オブジェクトの縦のサイズを指定します。

オブジェクトによっては高さが規定されているものがあり、その場合は変更できません。

JSON 文法

名称	データタイプ	とりうる値
height	number	最小値: 0

対象オブジェクト

4D View Pro エリア - 4D Write Pro エリア - ボタン - ボタングリッド - チェックボックス - コンボボックス - ドロップダウンリスト - グループボックス - 階層リスト - 入力 - リストボックス - 線 - リストボックス列 - 楕円 - ピクチャーボタン - ピクチャーポップアップメニュー - ブラグインエリア - 進歩インジケーター - ラジオボタン - ルーラー - 四角 - スピナー - スプリッター - スタティックピクチャー - ステッパー - サブフォーム - タブコントロール - テキストエリア - Web エリア

幅

オブジェクトの横のサイズを指定します。

- オブジェクトによっては高さが規定されているものがあり、その場合は変更できません。
- リストボックス列 に サイズ変更可 プロパティが設定されている場合には、ユーザーは手動でカラムサイズを変更することもできます。
- リストボックスの 横方向サイズ変更 プロパティに "拡大" を設定している場合にフォームをリサイズすると、一番右のカラムの幅は必要に応じて最大幅を超えて拡大されます。

JSON 文法

名称	データタイプ	とりうる値
width	number	最小値: 0

対象オブジェクト

4D View Pro エリア - 4D Write Pro エリア - ボタン - ボタングリッド - チェックボックス - コンボボックス - ドロップダウンリスト - グループボックス - 階層リスト - 入力 - リストボックス - 線 - リストボックス列 - 橋円 - ピクチャーボタン - ピクチャーポップアップメニュー - ブラグインエリア - 進捗インジケーター - ラジオボタン - ルーラー - 四角 - スピナー - スプリッター - ステイックピクチャー - ステッパー - サブフォーム - タブコントロール - テキストエリア - Web エリア

最大幅

列の最大幅 (ピクセル単位)。列やフォームをサイズ変更する際、このサイズよりも列幅を大きくすることはできません。

リストボックスの [横方向サイズ変更](#) プロパティに "拡大" を設定している場合にフォームをリサイズすると、一番右のカラムの幅は必要に応じて最大幅を超えて拡大されます。

JSON 文法

名称	データタイプ	とりうる値
maxWidth	number	最小値: 0

対象オブジェクト

[リストボックス列](#)

最小幅

列の最小幅 (ピクセル単位)。列やフォームをサイズ変更する際、このサイズよりも列幅を小さくすることはできません。

リストボックスの [横方向サイズ変更](#) プロパティに "拡大" を設定している場合にフォームをリサイズすると、一番右のカラムの幅は必要に応じて最大幅を超えて拡大されます。

JSON 文法

名称	データタイプ	とりうる値
minWidth	number	最小値: 0

対象オブジェクト

[リストボックス列](#)

行の高さ

リストボックス行の高さを設定します (ヘッダーおよびフッターは除きます)。デフォルトで、行の高さはプラットフォームとフォントサイズに基づき設定されます。

JSON 文法

名称	データタイプ	とりうる値
rowHeight	string	"em" または "px" (デフォルト) 単位の css 値

対象オブジェクト

[リストボックス](#)

参照

[行高さ配列](#)

行高さ配列

このプロパティは、リストボックスに関連付けたい行高さ配列の名前を指定するのに使用します。行高さ配列は数値型である必要があります (デフォルトは倍長整数)。

行高さ配列が定義されているとき、0 ではない値の要素はそれぞれ、リストボックスの対応する行の高さを決定する際に、選択されている行の高さ単位に基づいて考慮されます。

たとえば:

```
ARRAY LONGINT(RowHeights;20)
RowHeights{5}:=3
```

ここで行の単位が "行" であったとすると、リストボックスの 5 行目は 3 行分の高さになる一方、他の行はデフォルトの高さを保ちます。

- 行高さ配列プロパティは、階層リストボックスに対しては効力を持ません。
- 配列型のリストボックスの場合、このプロパティは [自動行高](#) オプションがチェックされていない場合に限り使用可能です。

JSON 文法

名称	データタイプ	とりうる値
rowHeightSource	string	4D 配列変数の名前

対象オブジェクト

[リストボックス](#)

参照

[行の高さ](#)

行列数

列

サムネールテーブルを構成する行数を指定します。

JSON 文法

名称	データタイプ	とりうる値
columnCount	integer	最小値: 1

対象オブジェクト

[ピクチャーボタン](#) - [ボタングリッド](#) - [ピクチャポップアップメニュー](#)

行

サムネールテーブルを構成する列数を指定します。

JSON 文法

名称	データタイプ	とりうる値
rowCount	integer	最小値: 1

対象オブジェクト

[ピクチャーボタン](#) - [ボタングリッド](#) - [ピクチャポップアップメニュー](#)

データソース

自動挿入

このオプションがチェックされていると、オブジェクトに関連付けられたリストにない値をユーザーが入力した場合に、その値が自動的にメモリー内のリストに追加されます。

自動挿入 のオプションが設定されていない場合、入力された値はフォームオブジェクトの中には保存されますが、メモリー内のリストには入力されません。

このプロパティは次のフォームオブジェクトでサポートされています:

- 選択リストと紐づけられている [コンボボックス](#) および [リストボックス列](#) フォームオブジェクト。
- 配列またはオブジェクトデータソースにより、紐づけられたリストが生成されている [コンボボックス](#) フォームオブジェクト。

たとえば、"France, Germany, Italy" という値を含む選択リストが "Countries" というコンボボックスに関連付けられていた場合を考えます。自動挿入 のオプションがチェックをされていて、ユーザーが "Spain" という値を入力すると、"Spain" という値が自動的にメモリー内のリストに追加されます:



デザインモードで定義された選択リストが関連付けられている場合、自動挿入によって、その元のリストが変更されることはありません。

JSON 文法

名称	データタイプ	とりうる値
automaticInsertion	boolean	true, false

対象オブジェクト

コンボボックス - リストボックス列

選択リスト

選択リストをフォームオブジェクトに関連づけます。指定できるのは選択リスト名 (リストの参照) またはデフォルト値のコレクションです。

選択リストをオブジェクトに紐づけるには、[OBJECT SET LIST BY NAME](#) または [OBJECT SET LIST BY REFERENCE](#) コマンドを使ってもおこなえます。

JSON 文法

名称	データタイプ	とりうる値
choiceList	リスト、コレクション	選択可能な値のリスト
list	リスト、コレクション	選択可能な値のリスト (階層リストのみ)

対象オブジェクト

ドロップダウンリスト - コンボボックス - 階層リスト - リストボックス列

選択リスト (静的リスト)

タブコントロールオブジェクトのラベルとして使用する静的な値のリスト。

JSON 文法

名称	データタイプ	とりうる値
labels	リスト、コレクション	タブコントロールラベルに使用する値のリスト。

対象オブジェクト

[タブコントロール](#)

カレントの項目

コレクションまたはエンティティセレクションリストボックス

ユーザーによって選択されたコレクション要素/エンティティが割り当てられる変数あるいは式を指定します。オブジェクト変数あるいはオブジェクトを受け入れる割り当て可能な式を使用する必要があります。ユーザーが何も選択しなかった場合、あるいはスカラー値のコレクションを使用した場合、Null 値が割り当てられます。

このプロパティは「読み取り専用」であり、リストボックスにおけるユーザーアクションに基づいて自動的に更新されます。この値を編集してリストボックスの選択状態を変更することはできません。

JSON 文法

名称	データタイプ	とりうる値
currentItemSource	string	オブジェクト型の式

対象オブジェクト

[リストボックス](#)

カレントの項目の位置

コレクションまたはエンティティセレクションリストボックス

ユーザーによって選択されたコレクション要素/エンティティの位置を表す倍長整数が割り当てられる変数あるいは式を指定します。

- 要素/エンティティが選択されていない場合、変数あるいは式は 0 を受け取ります。
- 単一の要素/エンティティが選択されている場合、変数あるいは式はその位置を受け取ります。
- 複数の要素/エンティティが選択されている場合、変数あるいは式は最後に選択された要素/エンティティの位置を受け取ります。

このプロパティは「読み取り専用」であり、リストボックスにおけるユーザーアクションに基づいて自動的に更新されます。この値を編集してリストボックスの選択状態を変更することはできません。

JSON 文法

名称	データタイプ	とりうる値
currentItemPositionSource	string	数値型の式

対象オブジェクト

データタイプ (式の型)

表示される式のデータタイプを定義します。このプロパティは次のフォームオブジェクトで使用されます:

- セレクションおよびコレクション型の [リストボックス列](#)。
- オブジェクトまたは配列と紐づいた [ドロップダウンリスト](#)。

[式タイプ](#) の章も参照ください。

JSON 文法

名称	データタイプ	とりうる値
dataSourceTypeHint	string	<ul style="list-style-type: none"> リストボックス列: "boolean", "number", "picture", "text", date", "time"。配列/セレクションリストボックスのみ: "integer", "object" ドロップダウンリスト: "object", "arrayText", "arrayDate", "arrayTime", "arrayNumber"

対象オブジェクト

オブジェクトまたは配列と紐づいた [ドロップダウンリスト - リストボックス列](#)

データタイプ (リスト)

[ドロップダウンリスト](#) に関連づけられたフィールドまたは変数に保存するデータの種類を定義します。このプロパティは次のフォームオブジェクトで使用されます:

- [選択リストと紐づいた ドロップダウンリスト](#)。
- [階層型の選択リストと紐づいた ドロップダウンリスト](#)

次の値が提供されています:

- リスト参照: ドロップダウンリストが階層型であることを宣言します。このドロップダウンリストは最大で 2つの階層レベルを表示することができ、その内容は Hierarchical Lists テーマの4Dランゲージコマンドで管理することができます。
- 選択された項目値 (デフォルト): ドロップダウンリストは階層型でなく、ユーザーによって選択された項目の値が直接保存されます。たとえば、ユーザーが "Blue" という値を選択した場合、この値がフィールドに保存されます。
- 選択された項目参照: ドロップダウンリストは階層型でなく、選択リスト項目の参照がオブジェクトに保存されます。この参照番号とは [APPEND TO LIST](#) または [SET LIST ITEM](#) コマンドの *itemRef* パラメーター、またはリストエディターを通してそれぞれの項目と関連付けされた数値です。このオプションにより、メモリーを節約することができます。フィールドに数値を保存するのは文字列を保存するより容量が軽いからです。また、これによりアプリケーションの翻訳が簡単になります。同じ項目の参照値を持つ、異なる言語で書かれた複数のリストを用意しておいて、アプリケーションの言語に応じたリストをロードするだけで多言語に対応できるからです。

選択された項目参照 オプションの使用の際には、以下の点に注意する必要があります:

- 参照を保存するには、データソースのフィールドまたは変数は、数値型である必要があります (リスト内に表示されている値の型とは関係ありません)。式の型 プロパティは自動的に設定されます。
- リストの項目には有効かつ固有の参照が関連付けられている必要があります。
- ドロップダウンリストはフィールドまたは変数と紐づいている必要があります。

JSON 文法

名称	データタイプ	とりうる値
saveAs	string	"value", "reference"

"type": "dropdown" フォームオブジェクトに "dataSourceTypeHint" : "integer" のみを設定すると、階層ドロップダウンリストが

宣言されます。

対象オブジェクト

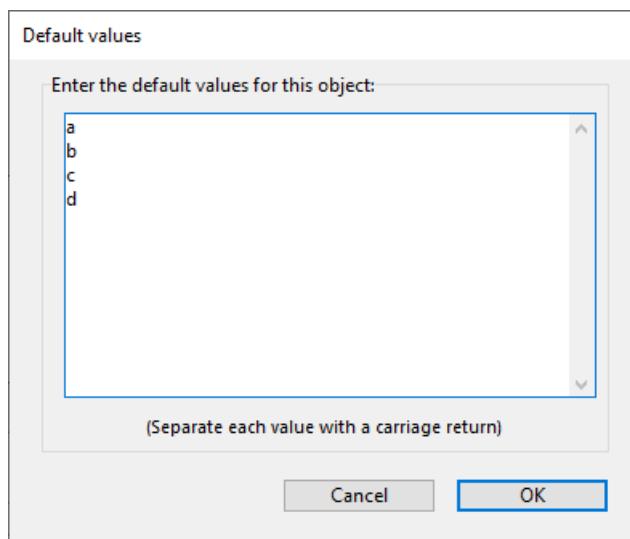
リストと紐づいた [ドロップダウンリスト](#)

デフォルト値

配列型リストボックスにおいて、リストボックス列のデフォルト値として使用される値のリストです。これらの値は自動で、フォームを実行したときにその列に割り当てられた [配列変数](#) に代入されます。この配列を参照することで、ランゲージを使ってオブジェクトを管理することができます。

このプロパティと、新規レコードのフィールド値を定義するのに使える入力オブジェクトの [デフォルト値](#) を混同しないようにしてください。

デフォルト値のリストを入力します。フォームエディター上で専用のダイアログが開き、改行で区切られた値を入力することができます。



リストボックス列に [選択リスト](#) を定義することもできます。選択リストは列の各行において選択可能な値の候補として使用されますが、デフォルト値のリストはカラムの各行に上から順に割り当てられます。

JSON 文法

名称	データタイプ	とりうる値
values	collection	デフォルト値 (文字列) のコレクション。例: "a", "b", "c", "d"

対象オブジェクト

[リストボックス列 \(配列型のみ\)](#)

式

[セレクション型](#) および [コレクション/エンティティセレクション型](#) リストボックスのプロパティです。 [変数あるいは式](#) の章も参照ください。

列に割り当てる 4D 式です。以下のものを指定できます:

- 単純な変数 (この場合、コンパイル用に明示的に型宣言されている必要があります)。BLOB と配列型以外のどんな型の変数も使用することができます。変数の値は通常 `On Display Detail` イベントで計算されます。
- 標準の [Table]Field シンタクスを使用した フィールド ([セレクション型リストボックス](#) のみ) 例: `[Employees]LastName`。以下の型のフィールドを使用できます:

- String
 - 数値
 - 日付
 - 時間
 - ピクチャー
 - ブール
- マスター テーブルおよび他のテーブルのフィールドを指定できます。
- 4D式 (単純な式、フォーミュラ、または 4Dメソッド)。式は値を返す必要があります。値は `On Display Detail` および `On Data Change` イベントで評価されます。式の結果は、アプリケーションモードでフォームを実行すると自動で表示されます。式は、セレクション型リストボックスではマスター テーブルの (カレントまたは命名) セレクションの各レコードごとに、コレクション型リストボックスではコレクションの各要素ごとに、エンティティセレクション型リストボックスではセレクションのエンティティごとに評価されます。空の場合、列には何も表示されません。
以下の型の式がサポートされています:
 - String
 - 数値
 - 日付
 - ピクチャー
 - ブール

コレクション/エンティティセレクション型リストボックスにおいては、Null あるいはサポートされない型は空の文字列として表示されます。
コレクションあるいはエンティティセレクションを使用する場合、カラムに割り当てられた要素プロパティ/エンティティ属性は、通常 `This` を含む式を用いて宣言します。この `This` は現在処理中の要素への参照を返す、専用の 4Dコマンドです。たとえば、`This.<propertyPath>` (ここでの`<propertyPath>` はコレクションのプロパティパス、あるいはエンティティ属性パス) を使用することで、各要素/エンティティのカレントの値にアクセスすることができます。
スカラー値のコレクションを使用した場合、4D は各コレクション要素に対して、単一のプロパティ (名前は "value") を持つオブジェクトを作成し、それに要素の値を格納します。この場合、`This.value` を式として使用します。

代入不可な式 (例: `[Person]FirstName+" "+[Person]LastName` など) を使用した場合、**入力可** オプションが選択されていても、その列に値を入力することはできません。

フィールド、変数、あるいは代入可能な式 (例: `Person.lastName`) を使用した場合、**入力可** プロパティの設定に基づき列への入力可/不可が決定されます。

JSON 文法

名称	データタイプ	とりうる値
dataSource	string	4D変数、フィールド名、あるいは任意のランゲージ式

対象オブジェクト

リストボックス列

マスター テーブル

カレントセレクションリストボックス

使用するカレントセレクションが属するテーブルを指定します。このテーブルとそのカレントセレクションが、リストボックスの列に割り当てられたフィールドの参照を形成します (フィールド参照やフィールドを含む式)。ある列が他のテーブルのフィールドを参照しているとしても、表示される行の数はマスター テーブルのカレントレコード数となります。

すべてのデータベーステーブルが利用できます。フォームがテーブルに属しているか (テーブルフォームの場合) あるいは属していないか (プロジェクトフォーム) は関係ありません。

JSON 文法

名称	データタイプ	とりうる値
table	number	テーブル番号

対象オブジェクト

リストボックス

関連付け

このプロパティは以下の場合に表示されます:

- オブジェクトに対して [選択リスト](#) が割り当てられている
- [入力](#) および [リストボックス列](#) の場合には、ユーザーがリスト内の値のみ入力できるように、オブジェクトに対して [指定リスト](#) も定義されている（通常は両方のオプションで同じリストを使用しているはずです）。

このプロパティは、選択リストに関連付けされたフィールドまたは変数において、フィールドに保存する内容の型を指定します:

- リスト項目の値（デフォルトのオプション）: ユーザーによって選択された項目の値が直接保存されます。たとえば、ユーザーが "Blue" という値を選択した場合、この値がフィールドに保存されます。
- リスト項目の参照番号: 選択リスト項目の参照がオブジェクトに保存されます。この参照番号とは [APPEND TO LIST](#) または [SET LIST ITEM](#) コマンドの *itemRef* パラメーター、またはリストエディターを通してそれぞれの項目と関連付けされた数値です。

このオプションにより、メモリーを節約することができます。フィールドに数値を保存するのは文字列を保存するより容量が軽いからです。また、これによりアプリケーションの翻訳が簡単になります。同じ項目の参照値を持つ、異なる言語で書かれた複数のリストを用意しておいて、アプリケーションの言語に応じたリストをロードするだけで多言語に対応できるからです。

リスト項目の参照番号の使用の際には、以下の点に注意する必要があります:

- 参照を保存するには、データソースのフィールドまたは変数は、数値型である必要があります（リスト内に表示されている値の型とは関係ありません）。[式の型](#) プロパティは自動的に設定されます。
- リストの項目には有効かつ固有の参照が関連付けられている必要があります。

JSON 文法

名称	データタイプ	とりうる値
saveAs	string	"value", "reference"

対象オブジェクト

入力 - リストボックス列

選択された項目

コレクションまたはエンティティセレクションリストボックス

ユーザーによって選択されている一つ以上のコレクション要素/エンティティが割り当てられる変数あるいは式を指定します。

- コレクションリストボックスにおいては、コレクション変数あるいはコレクションを受け入れる割り当て可能な式を使用する必要があります。
- エンティティセレクションリストボックスにおいては、エンティティセレクションオブジェクトがビルトされます。オブジェクト変数あるいはオブジェクトを受け入れる割り当て可能な式を使用する必要があります。

このプロパティは「読み取り専用」であり、リストボックスにおけるユーザーアクションに基づいて自動的に更新されます。この値を編集してリストボックスの選択状態を変更することはできません。

JSON 文法

名称	データタイプ	とりうる値
selectedItemsSource	string	コレクション式

対象オブジェクト

命名セレクション

命名セレクションリストボックス

使用する命名セレクションを指定します。有効な命名セレクションの名前を入力しなければなりません。使用できるのはプロセスあるいはインターフロセス命名セレクションです。リストボックスの内容はこのセレクションに基づきます。選択された命名セレクションは、リストボックスが表示される時点で存在し、有効でなければなりません。そうでない場合、リストボックスは空で表示されます。

命名セレクションはソート済みのレコードリストです。これはセレクション中のカレントレコードと並び順をメモリーに保持するために使用されます。
詳細は、4Dランゲージリファレンス マニュアルの 命名セレクション を参照してください。

JSON 文法

名称	データタイプ	とりうる値
namedSelection	string	命名セレクションの名前

対象オブジェクト

リストボックス

表示

文字フォーマット

文字フォーマットは、表示や印刷時に文字フィールドや変数にデータを表示する方法を制御します。以下は文字フィールド用に提供されるフォーマットのリストです：



このリストからフォーマットを選択するか、コンボボックスに入力することができます。フォーマットポップアップメニューには、主に使用される文字フォーマット（電話番号等）が用意されています。また、ツールボックスのフィルターとフォーマットで設定したカスタムフォーマットを選択することもできます。この場合、そのフォーマットをオブジェクトプロパティで変更することはできません。開発者が作成したカスタムフォーマットやフィルターはリストの先頭に表示されます。

シャープ (#) は文字表示フォーマットのプレースホルダーです。ハイフンやスペース、その他の句読点を表示したい場所に挿入できます。表示したい実際の句読点と、文字データを表示する場所には # を置きます。

たとえば部品番号が "RB-1762-1" のようなフォーマットの時、

文字フォーマットを以下のように書けます：

```
##-#####-#
```

ユーザーが "RB17621" と入力すると、フィールドには以下の通りに表示されます：

RB-1762-1

フィールドに実際に格納される値は "RB17621" です。

フォーマットが許可するよりも多くの文字が入力されると、4Dは最後の文字を表示します。たとえばフォーマットが以下の時：

```
(#####)
```

そしてユーザーが "proportion" と入力すると、フィールドには以下のように表示されます：

(portion)

フィールドには "proportion" が格納されます。表示フォーマットにかかわらず、4Dは入力された文字を受け入れ、格納します。データが失われることはありません。

JSON 文法

名称	データタイプ	とりうる値
textFormat	string	"### ####", "(##) ### ####", "### ### ####", "### ## ####", "00000", カスタムフォーマット

対象オブジェクト

日付フォーマット

日付フォーマットは、表示や印刷時に日付を表示する方法を制御します。データ入力の際は選択した表示フォーマットとは関係なく、YYYY/MM/DD 形式で日付を入力します。

[数値フォーマット](#) や [文字フォーマット](#) と異なり、日付表示フォーマットは4Dの組み込みフォーマットのなかから選択しなければなりません。

利用可能な日付表示フォーマットは以下のとおりです：

フォーマット	JSON 文字列	例
System date short	- (デフォルト)	20/03/25
System date abbreviated (1)	systemMedium	2020/03/25
System date long	systemLong	2020年3月25日 水曜日
RFC 822	rfc822	Tue, 25 Mar 2020 22:00:00 GMT
Short Century	shortCentury	03/25/20、ただし 04/25/2032 (2)
Internal date long	long	March 25, 2020
Internal date abbreviated (1)	abbreviated	Mar 25, 2020
Internal date short	short	03/25/2020
ISO Date Time (3)	iso8601	2020-03-25T00:00:00

(1) "June" は "Jun"、"July" は "Jul" に省略されます。

(2) 年は、1930年～2029年の間は2桁の数字で表示されますが、それ以外の場合は4桁で表示されます。これはデフォルト設定ですが、[SET DEFAULT CENTURY](#) コマンドで変更することができます。

(3) ISO Date Time フォーマットは XML の日付と時間表現の標準 (ISO8601) に対応します。これは主に XML フォーマットや Web サービスのデータを読み込んだり書き出したりするために使用します。

表示フォーマットにかかわらず、年度を2桁で入力すると、4D は年が00～29の間であれば 21世紀とみなし、30～99の間であれば 20世紀とみなします。これはデフォルト設定ですが、[SET DEFAULT CENTURY](#) コマンドで変更することができます。

JSON 文法

名称	データ タイプ	とりうる値
dateFormat	string	"systemShort", "systemMedium", "systemLong", "iso8601", "rfc822", "short", "shortCentury", "abbreviated", "long", "blankIfNull" (他の値と組み合わせることができます)

対象オブジェクト

数値フォーマット

数値フィールドには整数、倍長整数、整数64bit、実数、そしてフロート型が含まれます。

数値フォーマットは表示や印刷時に数値を表示する方法を制御します。選択した表示フォーマットとは関係なく、データ入力の際は数値だけを（必要に

応じ小数点やマイナス記号も) 入力します。

4Dは様々なデフォルトの数値表示フォーマットを提供しています。

プレースホルダー

それぞれの数値表示フォーマットでは、数値記号 (#)、ゼロ (0)、キャレット (^)、アスタリスク (*) をプレースホルダーとして使用します。表示しようとす
る各桁に対して 1つのプレースホルダーを使用し、独自の数値表示フォーマットを作成できます。

プレースホルダー	千項及び末尾のゼロ
#	何も表示しない
0	0を表示
^	スペースを表示 (1)
*	アスタリスクを表示

(1) キャレット (^) は、ほとんどのフォントの数字と同じ幅を占めるスペースを生成します。

たとえば、3桁の数字を表示する場合、###というフォーマットを使用できます。フォーマットにより許可された桁数を超えて入力すると、4Dは <<< を
フィールドに表示し、表示フォーマットで指定された桁数を超える入力がおこなわれたことを示します。

ユーザーがマイナスの数値を入力すると、左端の文字はマイナス記号として表示されます（負数の表示フォーマットが指定されていない場合）。##0とい
うフォーマットであれば、マイナス 26 は -26 と表示されます。マイナス260 は <<< と表示されますが、これはプレースホルダーが 3桁分しか指定されて
いないところに、マイナス記号により 1つのプレースホルダが使用されてしまい、桁あふれしたためです。

表示フォーマットとは関係なく、4Dはフィールドに入力された数値を受け入れ、保存します。データが失われることはありません。

各プレースホルダー文字は、先行のゼロや末尾のゼロを表示する上で、その効果に違いがあります。先行のゼロとは小数点より左側の数値の先頭にある
ゼロのことです。末尾のゼロは小数点より右側の数値の終わりにあるゼロのことです。

たとえば ##0 というフォーマットを使用して 3桁の数字を表示するものとします。ユーザーがフィールドに何も入力しないと、フィールドには 0 が表示され
ます。26 と入力すると、フィールドには 26 と表示されます。

区切り文字

数値表示フォーマット（科学的記数法を除く）は自動でシステムの地域パラメーターに基づきます。4D は OS に定義された小数点と千の位区切り文
字を使用して “.” と “,” 文字をそれぞれ置き換えます。0 や # に続くピリオドとコンマはプレースホルダー文字として扱われます。

Windows 環境下で、テンキーの小数点キーを使用した際、4Dはカーソルが位置しているフィールドの型に応じて挙動が変化します： * 実数
型のフィールドの場合、このキーを使用するとシステムによって定義された浮動小数点を挿入します。* それ以外の型のフィールドの場合、この
キーを使用するとそのキーに割り当てられた文字を挿入します。通常はピリオド (.) またはカンマ (,) です。

小数点とその他の表示文字

表示フォーマット内では 1つの小数点を使用することができます。ユーザーが小数点を入力するかどうかに関係なく、小数点を表示したい場合、ゼロの間
に小数点を置かなければなりません。

フォーマット内で他の文字を使用することもできます。文字を単独で使用したりプレースホルダーの前後に配置すると、その文字が常に表示されます。たと
えば次のようなフォーマットの場合：

¥##0

円記号はプレースホルダーの前に置かれているため、常に表示されます。

文字がプレースホルダーの間に置かれている場合、両側に数字が表示される場合のみ、その文字が表示されます。たとえばフォーマットを次のように指定
したとき：

```
###.##0
```

ポイント (点) は、ユーザが少なくとも4桁以上の数値を入力した場合にのみ表示されます。

数値表示フォーマットにおいて、スペースは文字として扱われます。

正数、負数、ゼロのフォーマット

数値表示フォーマットは最大で 3つの部分に分けられ、それぞれ正数、負数、ゼロの値に対応する表示フォーマットを指定できます。それぞれの部分は以下のように並び、セミコロンで区切られます：

```
正数;負数;ゼロ
```

3つの部分すべてを指定する必要はありません。1つの部分だけを使用する場合、4Dはすべての数値に対してそのフォーマットを使用し、負の数の先頭にマイナス記号を配置します。

2つの部分を指定する場合、4D は 1番目のフォーマットを正数とゼロに対して使用し、負数には 2番目のフォーマットを使用します。3つの部分をすべて指定すると、1番目のフォーマットを正数、2 番目を負数、3 番目をゼロに使用します。

3番目の部分 (ゼロ) は解釈されず、文字の置き換えをおこないません。###;###;# と指定した場合、ゼロ値は "#" と表示されます。言い換えると、表示フォーマットとして実際に指定されたものが、ゼロ値として表示されます。

次の数値表示フォーマットの例は、円記号とカンマを表示し、負の数値はカッコ内に入れ、ゼロを表示しません：

```
¥###,##0.00; ( ¥###,##0.00 );
```

2つ目のセミコロンにより、ゼロの表示には何も使用しないことを 4Dに指示している点に注目してください。次のフォーマットは前の例と似ていますが、2つ目のセミコロンが指定されていません。これにより、ゼロに対して正数のフォーマットを使用するよう 4Dに指示しています：

```
¥###,##0.00; ( ¥###,##0.00 )
```

この場合、ゼロは “¥0.00” と表示されます。

科学的記数法

科学的記数法で数値を表示したい場合には、アンパサンド (&) に続けて表示したい桁数を指定します。たとえば次のフォーマットを指定すると：

```
&3
```

759.62 は以下のように表示されます：

```
7.60e+2
```

科学的記数法フォーマットは、表示される数値を自動的に丸める唯一のフォーマットです。前述の例では、数値が 7.59e+2 と切り捨てられずに 7.60e+2 に丸められている点に注意してください。

16進フォーマット

次の表示フォーマットを使用して、数値を 16進表記で表示することができます：

- &x : このフォーマットでは 16進数が “0xFFFF” 形式で表示されます。
- &\$: このフォーマットでは 16進数が “\$FFFF” 形式で表示されます。

XML記法

&xml フォーマットを使用すると、数字を XML 標準ルールに沿ったものにします。特に小数点がシステム設定に関係なくすべての場合においてポイント(ピリオド)に変換されます。

数値を時間として表示する

&/ の後に数字を指定することにより、数値を時間として (時間フォーマットで) 表示することができます。時間は午前0 時を基点とした秒数として計算されます。フォーマット内の数字は表示フォーマットドロップダウンメニュー上でその時間フォーマットが表示される順番に相当します。

たとえば次のフォーマットを指定すると:

&/7

ドロップダウンメニューの7番目の時間フォーマット (AM/PM で表わす時間) に対応します。このフォーマットが指定された数値フィールドの場合、25000 は次のように表示されます:

6:56 AM

例題

次の表は各種フォーマットの数値表示への効果を表わしています。正数、負数、ゼロという 3つの欄では 1234.50、-1234.50、0 がそれぞれどのように表示されるかを示しています。

入力されたフォーマット	正数	負数	ゼロ
# ##	<<<	<<<	
# ## #	1234	<<<<	
# ## ## #	1234	-1234	
# ## ## .##	1234.5	-1234.5	
# ## ## 0.00	1234.50	-1234.50	0.00
# ## ## 0	1234	-1234	0
+### ## 0;-### ## 0;0	+1234	-1234	0
### ## 0DB;### ## 0CR;0	1234DB	1234CR	0
### ## 0;(### ## 0)	1234	(1234)	0
## ,## 0	1,234	-1,234	0
## ,## 0.00	1,234.50	-1,234.50	0.00
~~~~~	1234	-1234	
~~~~~^0	1234	-1234	0
~,^0	1,234	-1,234	0
~,^0.00	1,234.50	-1,234.50	0.00
*****	***1234	**-1234	*****
*****0	***1234	**-1234	*****0
, **0	**1,234	*-1,234	***0
*, **0.00	*1,234.50	-1,234.50	*****0.00
\$*, **0.00;-\$*, **0.00	\$1,234.50	-\$1,234.50	\$*****0.00
\$~~~~~0	\$ 1234	\$-1234	\$ 0
\$~~~0;-\$~~~0	\$1234	-\$1234	\$ 0
\$~~~0 ;(\$~~~0)	\$1234	(\$1234)	\$ 0
\$~,^0.00 ;(\$~,^0.00)	\$1,234.50	(\$1,234.50)	\$ 0.00
&2	1.2e+3	-1.2e+3	0.0e+0
&5	1.23450e+3	-1.23450e+3	0.00000
&xml	1234.5	-1234.5	0

JSON 文法

名称	データタイプ	とりうる値
numberFormat	string	数値 (必要に応じて小数点およびマイナス記号を含む)

対象オブジェクト

コンボボックス - ドロップダウンリスト - 入力 - リストボックス列 - リストボックスフッター - 進捗インジケーター

ピクチャーフォーマット

ピクチャーフォーマットはピクチャーが表示あるいは印刷される際の表示方法を制御します。データ入力時はフォーマットに関わらず、ユーザーはクリップボードからのペーストやドラッグ & ドロップでピクチャーを入力します。

トランケートとスケーリングオプションを選択しても、ピクチャーが変更されることではなく、ピクチャーフィールドのデータは失われません。ピクチャー表示フォーマットはピクチャーの表示にのみ影響します。

スケーリング

JSON 文法では: "scaled"

スケーリングを選択すると、ピクチャーはフィールドエリアの大きさに合うようにリサイズされます。



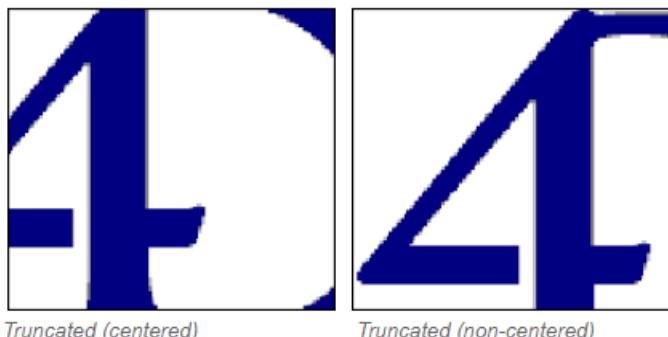
トランケート (中央合わせ/中央合わせしない)

JSON 文法では: "truncatedCenter" / "truncatedTopLeft"

トランケート (中央合わせ) フォーマットを選択すると、4D はエリアの中央にピクチャーを配置し、収まらない部分はエリアからはみ出します。上下、および左右のはみ出し量は同じになります。

トランケート (中央合わせしない) フォーマットを選択すると、4D はピクチャーの左上角をフィールドの左上角に合わせて配置し、フィールドエリアに収まらない部分はエリアからはみ出します。ピクチャーは右と下にはみ出します。

ピクチャーフォーマットが トランケート (中央合わせしない) の場合、入力エリアにスクロールバーを追加できます。



スケーリング (プロポーショナル) とスケーリング (中央合わせ・プロポーショナル)

JSON 文法では: "proportionalTopLeft" / "proportionalCenter"

スケーリング (プロポーショナル) を使用すると、ピクチャーエリアに収まるよう、比率を保ったままサイズが調整されます。スケーリング (中央合わせ・プロポーショナル) オプションも同様ですが、ピクチャーはエリアの中央に配置されます。

ピクチャーがエリアよりも小さい場合、サイズは変更されません。ピクチャーがエリアよりも大きい場合、そのエリア内に全体が表示されるよう、比率を保ったままサイズが小さくなります。比率が保たれるため、ピクチャーは歪むことなく表示されます。

中央合わせを選択した場合、画像はエリアの中央に配置されます:



Scaled to fit (proportional)

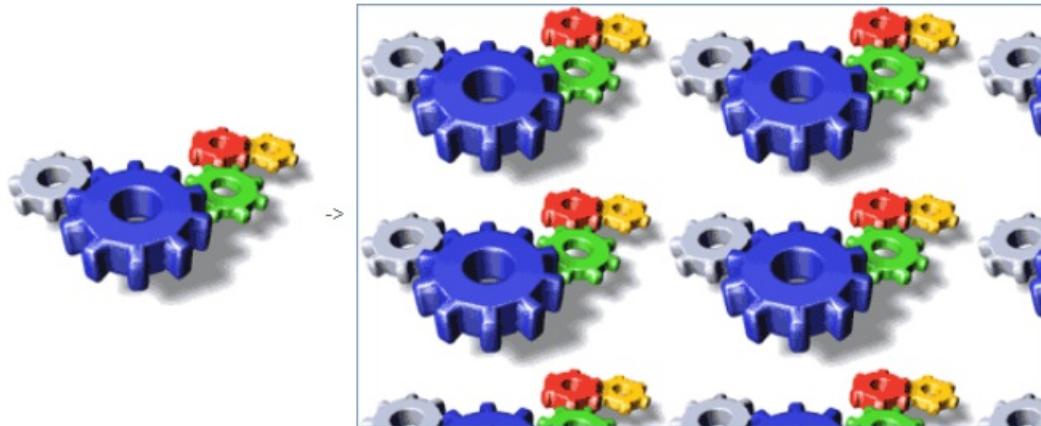


Scaled to fit centered (proportional)

繰り返し

JSON 文法では: "tiled"

繰り返し フォーマットを持つピクチャーが含まれるエリアが拡大されると、ピクチャーは変形されず、エリア全体を埋めるのに必要なだけピクチャーが繰り返されます。



フィールドがオリジナルのピクチャーよりも小さいサイズにされた場合、ピクチャーはトランケート（中央合わせなし）されます。

JSON 文法

名称	データタイプ	とりうる値
pictureFormat	string	"truncatedTopLeft", "scaled", "truncatedCenter", "tiled", "proportionalTopLeft", "proportionalCenter"

対象オブジェクト

入力 - リストボックス列 - リストボックスフッター

時間フォーマット

時間フォーマットは、表示や印刷時に時間を表示する方法を制御します。選択した表示フォーマットとは関係なく、データ入力の際は 24 時間制の “HH:MM:SS” フォーマット、または 12 時間制の “HH:MM:SS AM/PM” フォーマットで時間を入力します。

文字 や 数値 の表示フォーマットとは異なり、時間の表示フォーマットはフォーマットポップアップメニューから選択しなければなりません。

次の表は、時間フィールドの表示フォーマットとそれぞれのフォーマットの例を示しています：

フォーマット	JSON 文字列	コメント	04:30:25 の例
HH:MM:SS	hh_mm_ss		04:30:25
HH:MM	hh_mm		04:30
Hour Min Sec	HH_MM_SS		4 時 30 分 25 秒
Hour Min	HH_MM		4 時 30 分
HH:MM AM/PM	hh_mm_am		4:30 AM
MM SS	mm_ss	00:00:00からの経過時間	270:25
Min Sec	MM_SS	00:00:00からの経過時間	270 分 25 秒
ISO Date Time	iso8601	時間に関する XML 標準表現に対応。主に XML フォーマットでのデータのやり取りに使用します。	0000-00-00T04:30:25
System time short	- (デフォルト)	システムに定義された標準の時間フォーマット	04:30:25
System time long abbreviated	systemMedium	macOSのみ: システムに定義された時間フォーマットの短縮型。 Windows では System time short フォーマットと同じ	04:30:25
System time long	systemLong	macOSのみ: システムに定義された時間フォーマット。 Windows では System time short フォーマットと同じ	04:30:25 JST

JSON 文法

名称	データタイプ	とりうる値
timeFormat	string	"systemShort", "systemMedium", "systemLong", "iso8601", "hh_mm_ss", "hh_mm", "hh_mm_am", "mm_ss", "HH_MM_SS", "HH_MM", "MM_SS", "blankIfNull" (他の値と組み合わせることができます)

対象オブジェクト

[コンボボックス](#) - [ドロップダウンリスト](#) - [入力](#) - [リストボックス列](#) - [リストボックスフッター](#)

テキスト (True時)/テキスト (False時)

ブール式 を次のフォームオブジェクトで表示した場合:

- [入力オブジェクト](#) にテキストとして
- [リストボックス列](#) に表示タイプ "ポップアップ" を選択して

... 値の代わりに表示するテキストを指定することができます:

- テキスト (True時) - 値が "true" の時に表示するテキスト
- テキスト (False時) - 値が "false" の時に表示するテキスト

JSON 文法

名称	データタイプ	とりうる値
booleanFormat	string	

対象オブジェクト

[リストボックス列](#) - [入力](#)

表示タイプ

列のデータに表示フォーマットを割り当てるために使用します。提供されるフォーマットは変数型（配列型のリストボックス）またはデータ/フィールド型（セレクションおよびコレクション型のリストボックス）により異なります。

ブール式および数値（数値または整数）式の列はチェックボックスとして表示することができます。表示タイプにチェックボックスを選択すると、[タイトル](#)プロパティが表示され、チェックボックスのタイトルを設定できます。

ブール式の列はポップアップメニューとしても表示することができます。この場合には、[テキスト \(True時\)](#) と [テキスト \(False時\)](#) プロパティが表示され、ポップアップメニューの対応するタイトルを設定できます。

JSON 文法

名称	データタイプ	とりうる値
controlType	string	<ul style="list-style-type: none">数値列: "automatic"（デフォルト）または "checkbox"ブール列: "checkbox"（デフォルト）または "popup"

対象オブジェクト

[リストボックス列](#)

レンダリングしない

このプロパティが選択されていると、アプリケーションモードでオブジェクトが描画されません（あとから描画することができます）。

このプロパティは、"透明" ボタンの実装を可能にします。レンダリングされていないボタンは、描画オブジェクトの上に配置することができます。レンダリングされていないボタンは、クリックされてもハイライトされることなく非表示のままで、クリックによるイベントは発生します。

JSON 文法

名称	データタイプ	とりうる値
display	boolean	true, false

対象オブジェクト

[ボタン - ドロップダウンリスト](#)

スリーステート

チェックボックスオブジェクトに、3 番目の状態を付加します。チェックボックスが 3番目の状態になると、チェックボックスに関連付けられた変数は値2を返します。

リストボックス列におけるスリーステートチェックボックス

[式タイプ](#) が数値型のリストボックス列は、スリーステートチェックボックスとして表示できます。スリーステートチェックボックスタイプを選択すると、以下の値が表示されます：

- 0 = チェックされていない
- 1 = チェックされている
- 2 (または2以上の任意の数値) セミチェックボックス（三番目の状態）データ入力時、この状態は2を返します。
- 1 = 非表示チェックボックス
- 2 = チェックされていない、入力不可
- 3 = チェックしている、入力不可
- 4 = セミチェックボックス、入力不可

スリーステートチェックボックスの場合も、[タイトル](#)プロパティが表示され、チェックボックスのタイトルを設定できます。

JSON 文法

名称	データタイプ	とりうる値
threeState	boolean	true, false

対象オブジェクト

[チェックボックス - リストボックス列](#)

タイトル

タイトルは、リストボックス列のプロパティとして次の場合に提供されます:

- 式タイプがブールで表示タイプが"チェックボックス"の場合
- 式タイプが数値(数値または整数)で表示タイプが"スリーステートチェックボックス"の場合

これらの場合に、チェックボックスのタイトルをこのプロパティで設定できます。

JSON 文法

名称	データタイプ	とりうる値
controlTitle	string	タイトル用のあらゆる文字列

対象オブジェクト

[リストボックス列](#)

エリプシスを使用して省略

リストボックスのカラムが、中身をすべて表示するには狭すぎる場合の値の表示を管理します。

このオプションは、どのような型の中身に対しても利用可能です(ただしピクチャーとオブジェクトを除く)。

- このオプションがチェックされているとき(デフォルト)、リストボックスセルの中身がカラムの幅を超えた場合、それらは省略され、エリプシスが表示されます:

Cities	Popu...
Charlotte	792862
New York	8406...
Philadelph...	1553...
San Fran...	837442
San Jose	288054

エリプシスの位置はOSによって変わります。上記の例(Windows)では、テキストの右側に表示されます(テキストの後半が省略されます)。macOS上では、テキストの真ん中に表示されます(テキストの中盤が省略されます)。

- このオプションのチェックが外れているとき、セルの中身がカラムの幅を超えていた場合、収まりきらない部分は表示されず、エリプシスも表示されません:

Cities	Popu...
Charlotte	792862
New York	3406000
Philadelphia	1553000
San Francisco	837442
San Jose	288054

エリプシスで省略オプションはデフォルトではチェックされており、配列、セレクション、コレクション型のリストボックスに対して指定可能です。

テキストまたは文字列型のカラムに対して適用した場合、エリプシスで省略オプションは **ワードラップ** オプションがチェックされていない場合にのみ使用可能です。ワードラップオプションがチェックされていた場合、セル内を超えたコンテンツについてはワードラップ機能によって管理されますので、エリプシスで省略オプションは使用できません。

エリプシスで省略オプションはブール型のカラムに対しても適用可能です。しかしながら、[セルのフォーマット](#)によって表示される結果は異なります：

- 表示タイプがポップアップに設定されている場合は、エリプシスでラベルが省略されます。
- 表示タイプがチェックボックスに設定されている場合は、ラベルは常に見切れます（エリプシスで省略されません）。

JSON 文法

名称	データタイプ	とりうる値
truncateMode	string	"withEllipsis", "none"

対象オブジェクト

[リストボックス列 - リストボックスヘッダー](#)

表示状態

このプロパティが選択されていると、アプリケーションモードでオブジェクトが非表示になります。

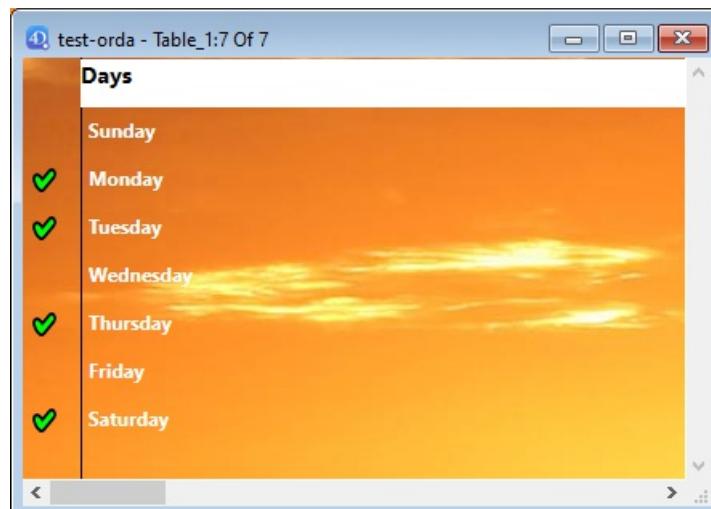
大部分のオブジェクトに対して、表示状態プロパティを指定することができます。このプロパティは主に、ダイナミックなインターフェースの開発を容易にするために使用されます。インターフェースを開発するとき、多くの場合はフォームの `On Load` イベント中にプログラムからオブジェクトを非表示にした後で、一部のオブジェクトを再度表示する必要性が頻繁に生じます。表示状態プロパティを使用すると、特定オブジェクトをあらかじめ非表示にしておくことにより、このロジックを逆に働かせることができます。この後、必要に応じて `OBJECT SET VISIBLE` コマンドを使用し、これらのオブジェクトを表示するようプログラミングすることができます。

リストフォームにおける自動表示

"リスト" フォーム のコンテキストにおいては、表示状態は次の二つの値をサポートします：

- レコード選択時 (JSON名: "selectedRows")
- レコード非選択時 (JSON名: "unselectedRows")

これらのプロパティ値は、リストフォームのボディに配置されたオブジェクトを描画する場合にのみ使用されます。具体的には、処理中のレコードが選択されているかいないかに応じて、当該オブジェクトを描画するかどうかを 4D に指示します。これにより、ハイライト以外の視覚的属性でもって、レコードの選択を表現することができます：



オブジェクトが `OBJECT SET VISIBLE` コマンドで非表示にされた場合、4D はこのプロパティを無視します。つまり、レコードの選択状態にかかわらず、当該オブジェクトは非表示のままになります。

JSON 文法

名称	データタイプ	とりうる値
visibility	string	"visible", "hidden", "selectedRows" (リストフォームのみ), "unselectedRows" (リストフォームのみ)

対象オブジェクト

4D View Pro エリア - 4D Write Pro エリア - ボタン - ボタングリッド - チェックボックス - コンボボックス - ドロップダウンリスト - グループボックス - 階層リスト - リストボックス - リストボックス列 - リストボックスフッター - リストボックスヘッダー - ピクチャーボタン - ピクチャーポップアップメニュー - プラグインエリア - 進捗インジケーター - ラジオボタン - スピナー - スプリッター - スタイックピクチャ - ステッパー - サブフォーム - タブコントロール - テキストエリア - Web エリア

ワードラップ

入力 オブジェクトにおいては、**複数行** プロパティが "はい" に設定されている場合にのみ、このプロパティは表示されます。

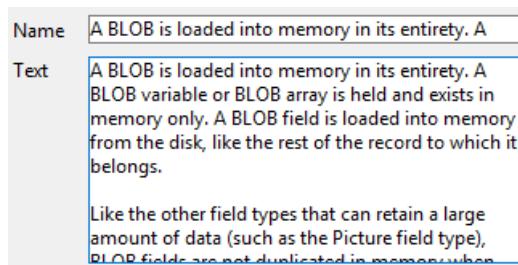
このオプションは、表示する内容がオブジェクトの幅を超えたときの表示を管理します。

リストボックスにてチェック / 入力オブジェクトで "はい" に設定

JSON 文法では: "normal"

このオプションがチェックされていると、テキストがカラムやエリアの幅を越えたときに、カラムやエリアの高さが許容する範囲内で自動的に次の行へと改行します。

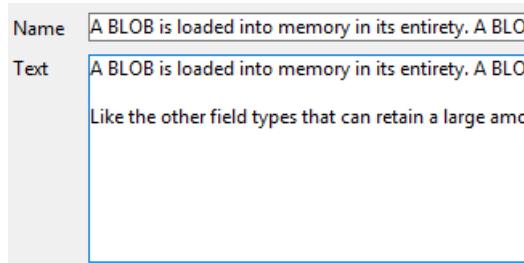
- 一行のカラムやエリアの場合、全体が表示できる最後の単語までが表示されます。4Dは改行を挿入します。下矢印キーを押すことで、エリアの内容をスクロールできます。
- 複数行のカラムやエリアの場合、4Dは自動改行を実行します。



リストボックスにてチェックなし / 入力オブジェクトで "いいえ" に設定

JSON 文法では: "none"

このオプションの場合、4D はいつさい自動改行をおこないません。表示可能な最後の単語はエリアをはみ出します。テキストタイプのエリアでは改行がサポートされます:



リストボックスの場合、長すぎるテキストは切り落とされ、省略記号 (...) が表示されます。以下の例では、左の列ではワードラップのオプションがチェックされていて、右の列ではされていません:

Header1	Header2
The vertical alignment will be applied as long as the full text fits in the cell. Otherwise, the text will be aligned to the top so as the beginning of the text will visible.	The vertical alignment will be ap...
You can make a field invisible in the Application environment and for the plug-ins by selecting the Invisible property for this field.	You can make a field invisible in ...

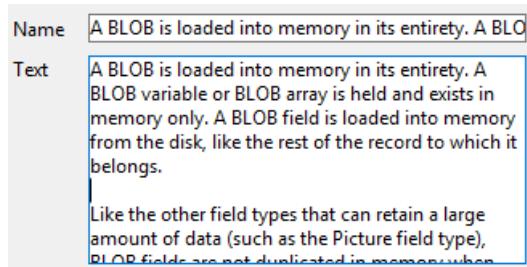
ワードラップの オプションの値に関わらず、行の高さは変化しないことに注意してください。改行を含むテキストがカラムの中に表示しきれないとき、表示しきれない部分は 切り落とされ、省略記号も表示されません。単一の行を表示するリストボックスの場合、テキストの最初の行のみ表示されます:

Header1	Header2
The vertical alignment will be	The vertical alignment will be ap...
You can make a field invisible in	You can make a field invisible in ...

入力オブジェクトで "自動" に設定 (デフォルト)

JSON 文法では: "automatic"

- 一行のエリアの場合、行の最後に表示される単語は切り落とされ、改行はされません。
- 複数行のエリアの場合、4Dは自動改行を実行します。



JSON 文法

名称	データタイプ	どううる値
wordwrap	string	"automatic" (リストボックスを除く), "normal", "none"

対象オブジェクト

[入力 - リストボックス列 - リストボックスフッター](#)

入力

自動スペルチェック

4D にはカスタマイズ可能なスペルチェック機能がビルトインされています。スペルチェックはテキスト型の [入力](#) オブジェクト、そして [4D Write Pro](#) ドキュメントに対して実行可能です。

自動スペルチェックプロパティは、各オブジェクトのスペルチェックを有効にします。この場合、スペルチェックはタイプ中に自動的に実行されます。チェックしたいオブジェクトそれぞれに対して `SPELL CHECKING` 4D ランゲージコマンドを呼び出して実行することもできます。

JSON 文法

名称	データタイプ	とりうる値
spellcheck	boolean	true, false

対象オブジェクト

[4D Write Pro エリア](#) - [入力](#)

コンテキストメニュー

このプロパティを有効にすると、フォームの実行中にオブジェクトに対して標準のコンテキストメニューが使用できるようになります。

ピクチャー型の [入力](#) オブジェクトの場合、標準の編集コマンド（カット、コピー、ペースト、そしてクリア）に加え、ファイルからピクチャーを読み込むために使用することができる 読み込み... コマンド、ピクチャーをディスクに保存するのに使用する 別名で保存... コマンドなどがあります。また、メニューを使用してピクチャーの表示フォーマットを変更することもできます。トランケート（中央合わせなし）、スケーリング そして スケーリング（中央合わせ/プロポーショナル）から選択できます。このメニューを使用した [表示フォーマット](#) の変更は一時的なものであり、レコードには保存されません。

[マルチスタイル](#) オプションがチェックされているテキスト型の [入力](#) オブジェクトの場合、標準の編集コマンド以外に以下の様なコマンドを使用することができます：

- フォント...: フォントシステムダイアログボックスを表示させます。
- 最近使用したフォント: セッション中に最近使用されたフォント名を表示します。リストには最大で 10 フォントまで表示されます（それ以上は古いものから置き換えられていきます）。デフォルトではリストは空になっているので、このオプションは表示されません。このリストは `SET RECENT FONTS` と `FONT LIST` コマンドを使用して管理することができます。
- スタイルの変更をおこなうためのコマンド: スタイル、サイズ、カラー、背景色。このポップアップメニューを使用してユーザーがスタイル属性を編集する、4D は `On After Edit` フォームイベントを生成します。

[Webエリア](#) の場合、メニューの内容はプラットフォームの描画エンジンにより設定されます。コンテキストメニューへのアクセスは `WA SET PREFERENCE` コマンドを使用して制御できます。

JSON 文法

名称	データタイプ	とりうる値
contextMenu	string	"automatic" (省略時のデフォルト), "none"

対象オブジェクト

[入力](#) - [Webエリア](#) - [4D Write Pro エリア](#)

入力可

入力可属性は、ユーザーがオブジェクトに値を入力できるかどうかを指定します。

すべてのアクティプオブジェクトはデフォルトで入力可です。フォーム上の特定のフィールドやオブジェクトを入力不可にしたい場合、入力可チェックボックスの選択を解除します。入力不可のオブジェクトはデータの表示のみをおこないます。当該フィールド名や変数名を使用するメソッドによってデータを制御します。入力不可オブジェクトでも `On Clicked`, `On Double Clicked`, `On Drag Over`, `On Drop`, `On Getting Focus` そして `On Losing Focus` フォームイベントは使用できます。これらによって、カスタムコンテキストメニューの管理が容易になり、入力不可変数をドラッグ & ドロップしたり選択したりできるインターフェースをデザインすることができます。

このプロパティを無効にした場合、リストによってリストボックス列に関連付けられたポップアップメニューも使用できなくなります。

JSON 文法

名称	データタイプ	とりうる値
enterable	boolean	true, false

対象オブジェクト

[4D Write Pro エリア](#) - [チェックボックス](#) - [階層リスト](#) - [入力](#) - [リストボックス列](#) - [進捗インジケーター](#) - [ルーラー](#) - [ステッパー](#)

入力フィルター

日本語利用時の注意点: 入力フィルターは日本語IMEと互換性がありません。入力文字種の制限及び#を使用した入力文字数の制限もできません。たとえば半角数字のみを2文字だけ入力を許可する目的で、入力フィルターに&9# #と指定しても、IME経由での全角数字やその他日本語文字の入力を防ぐことはできませんし、任意の数の文字が入力できてしまいます。アプリケーション仕様としてこのような制御が必要な場合は4Dコマンドを使用する必要があります。

入力フィルターを使用するとデータ入力中にユーザーがタイプできる文字を制御できます。[指定リスト](#)とは異なり、入力フィルターは文字ごとに処理がおこなわれます。たとえば、バーツ番号が常に2つの文字とそれに続く3つの数字で構成されるとき、入力フィルターを通してそのパターンを強制することができます。さらに特定の文字や数字のみを使用するよう制御することもできます。

入力フィルターはデータ入力時にのみ動作します。オブジェクトの選択をユーザーが解除した後のデータ表示には効果がありません。通常は、入力フィルターを[表示フォーマット](#)と一緒に使用します。フィルターはデータ入力を制約し、表示フォーマットはデータ入力後の値の表示を制御します。

データ入力中、タイプされたたびに入力フィルターは文字を評価します。ユーザーが無効な入力をすると(たとえば文字の代わりに数字)、4Dはその入力を受け付けません。ユーザーが有効な入力をおこなうまで値は変更されません。

入力フィルターに表示フォーマットを併用することで、形式的な文字をユーザーが入力しなくてすむようにできます。たとえば、アメリカ合衆国の電話番号は3桁のエリアコードに、3桁と4桁に分割される7桁の番号が続きます。エリアコードをカッコでくくり、電話番号の3つ目の数字の後にダッシュを表示するような表示フォーマットを利用することができます。このようなフォーマットが指定されている場合、カッコやダッシュをユーザーが入力する必要はありません。

入力フィルターの定義

ほとんどの場合、あらかじめ用意されている4Dの[ビルトインフィルター](#)を使用することができます。しかし、カスタマイズされたフィルターを作成することも可能です:

- 入力フィルターコードを直接入力することができます。
- ツールボックスのフィルターエディターで入力フィルターを作成し、その名前を指定することもできます。開発者が作成したカスタムフィルターはリストの先頭に表示されます。

入力フィルターの作成に関する詳細は[フィルターとフォーマットのコード](#)を参照ください。

デフォルト入力フィルター

入力フィルタードロップダウンリストから選択できる入力フィルターの説明は以下の表の通りです:

入力フィルター	説明
~A	すべての文字が入力可能、ただし大文字に変換されます。
&9	数字のみ入力可能。
&A	大文字の文字だけが入力可能。
&a	文字だけが入力可能(大文字と小文字)。
&@	数字と文字が入力可能。特殊記号を除きます。
~a##	2桁の任意の文字が入力可能、大文字に変換されます。(アメリカ合衆国の州名などに使われます)
!0&9##/#/#/#	標準の日付入力フォーマット。入力領域に0を表示します。任意の数値が入力可能。
!0&9##年##月##日	カスタム日付入力フォーマット。入力領域に0を表示します。任意の数値が入力可能。年月日が2桁ずつ入力可能。
!0&9##:#:#	時間入力フォーマット。時と分だけが入力可能。入力領域に0を表示します。任意の4桁の数字が入力可能。
!0&9##時##分	時間入力フォーマット。入力領域に0を表示します。時間と分数を2桁ずつ入力可能。
!0&9##時##分##秒	時間入力フォーマット。入力領域に0を表示します。時間と分数、秒数を2桁ずつ入力可能。
!0&9## #-## #-##	ローカルな郵便番号フォーマット。入力領域に0を表示します。任意の数値が入力可能。(3桁の数字と4桁の数字)
!_&9(##) !0## #-## ##	長距離電話番号フォーマット。先頭の3桁の入力領域はカッコで囲み(空の場合はアンダースコアを表示し)、残りの入力領域に0を表示。
!0&9## #-## #-##	長距離電話番号フォーマット。入力領域に0を表示します。任意の数値が入力可能。3桁と3桁と4桁の数字をハイフンで区分。
!0&9## #-## #-##	アメリカ合衆国のお年玉番号フォーマット。入力領域に0を表示します。任意の数値が入力可能。
~"A-Z;0-9; ;;.;-"	大文字の文字と句読点。大文字の文字、数字、スペース、コンマ、ピリオド、ハイフンだけが入力可能。
&"a-z;0-9; ;;.;-"	大文字と小文字の文字と句読点。大小の文字、数字、スペース、コンマ、ピリオド、ハイフンだけが入力可能。
&"0-9;.;-"	数字、小数点、ハイフン(マイナス記号)だけが入力可能。

JSON 文法

名称	データタイプ	とりうる値
entryFilter	string	<ul style="list-style-type: none"> 入力フィルターコード、または 入力フィルター名

対象オブジェクト

[チェックボックス](#) -

[コンボボックス](#) - [階層リスト](#) - [入力](#) - [リストボックス列](#)

フォーカス可

オブジェクトに対し フォーカス可 プロパティが選択されていると、そのオブジェクトはフォーカスを得ることができ、キーボードなどを使用してアクティビ化することができます。オブジェクトはフォーカスを得ると、オブジェクトごとあるいは OS ごとに定められた方法でハイライトされます。ただし [フォーカスの四角を隠す](#) オプションが選択されている場合を除きます。

[入力可](#) に設定された [入力オブジェクト](#) は常にフォーカス可です。

- [Current Member](#)

選択時にフォーカスを表示しているチェックボックス

- Current Member

選択されているが、フォーカスを表示していないチェックボックス

入力できないオブジェクトに フォーカス可 プロパティが設定されていると、ユーザーはエリアの内容を選択、コピー、およびドラッグ & ドロップすることができます。

JSON 文法

名称	データタイプ	とりうる値
focusable	boolean	true, false

対象オブジェクト

[4D Write Pro エリア](#) - [ボタン](#) - [チェックボックス](#) - [ドロップダウンリスト](#) - [階層リスト](#) - [入力](#) - [リストボックス](#) - [プラグインエリア](#) - [ラジオボタン](#) - [サブフォーム](#)

キーボードレイアウト

このプロパティは [入力](#) オブジェクトに対して特定のキーボードレイアウトを関連付けます。たとえば、国際的なアプリケーションにおいて、フォーム内にギリシャ文字で入力しなければならないフィールドがあった場合、"ギリシャ語" のキーボードレイアウトをこのフィールドに対して関連付けることができます。これにより、このフィールドがフォーカスを受けている場合にはデータ入力時にキーボード設定が自動的に変わります。

デフォルトでは、オブジェクトはカレントのキーボードレイアウトを使用します。

このプロパティは、`OBJECT SET KEYBOARD LAYOUT` と `OBJECT Get keyboard layout` コマンドを使用して動的に設定することができます。

JSON 文法

名称	データタイプ	とりうる値
keyboardDialect	テキスト	言語コード (例: "ar-ma", "cs" など) RFC3066, ISO639 および ISO3166 を参照ください。

対象オブジェクト

[4D Write Pro エリア](#) - [入力](#)

複数行

このプロパティは、テキストタイプの式や、文字およびテキストタイプのフィールドが割り当てられている [入力オブジェクト](#) で使用できます。値は、あり・なし・自動 (デフォルト) が選択できます。

自動

- 一行の入力オブジェクトでは、行の最後にある単語はエリアからはみ出し、改行はおこなわれません。
- 複数行の入力オブジェクトの場合、4D は自動で改行します:

Name	A BLOB is loaded into memory in its entirety. A BLO
Text	A BLOB is loaded into memory in its entirety. A BLOB variable or BLOB array is held and exists in memory only. A BLOB field is loaded into memory from the disk, like the rest of the record to which it belongs. Like the other field types that can retain a large amount of data (such as the Picture field type), BLOB fields are not duplicated in memory when

×

- 一行の入力オブジェクトでは、行の最後にある単語はエリアからはみ出し、改行はおこなわれません。
- 改行はおこなわれません。テキストは常に一行で表示されます。文字やテキストのフィールドまたは変数が改行文字を含んでいる場合、エリアが更新されるとすぐに最初のキャリッジターンより後のテキストが取り除かれます：

Name	A BLOB is loaded into memory in its entirety. A BLO
Text	A BLOB is loaded into memory in its entirety. A BLO

○

この値を選択すると、追加の [ワードラップ](#) オプションが表示されます。

JSON 文法

名称	データタイプ	とりうる値
multiline	テキスト	"yes", "no", "automatic" (定義されていない場合のデフォルト)

対象オブジェクト

入力

プレースホルダー

4D では、フォームのフィールド内にプレースホルダーテキストを表示することができます。

このテキストはフィールド内で半透明のテキストとして表示され、入力されるデータに関するヘルプ、指示、具体例などを表示します。このテキストは、ユーザーが文字をエリアに入力した瞬間に表示されなくなります：

Product Number	Enter 12-digit product code
Product Number	1

プレースホルダーテキストは、フィールドの中身が消去されると再び表示されます。

プレースホルダーとして表示できるデータの型は以下の通りです：

- 文字列 (テキストまたは文字)
- 日付または時刻 (ヌルのときブランクにする のプロパティがチェックされている場合に限ります)

xliff 参照を ":xliff:resname" の形でプレースホルダーとして使用することもできます。たとえば：

```
:xliff:PH_Lastname
```

この場合、"プレースホルダー" のフィールドには参照のみを渡します。参照と静的なテキストを組み合わせることはできません。

プレースホルダーのテキストは、[OBJECT SET PLACEHOLDER](#) と [OBJECT Get placeholder](#) コマンドを使って、プログラミングによって設定したり取得したりすることができます。

JSON 文法

名称	データタイプ	とりうる値
placeholder	string	オブジェクトに値が格納されていない場合に表示する半透明のテキスト

対象オブジェクト

[コンボボックス - 入力](#)

参照

ヘルプTips

選択を常に表示

このプロパティを選択すると、オブジェクト中で選択した文字列の反転状態が、フォーカスを失った後も表示されるようになります。これにより、テキストスタイルを更新するようなインターフェースの実装が容易になります ([マルチスタイル 参照](#))。

JSON 文法

名称	データタイプ	とりうる値
showSelection	boolean	true, false

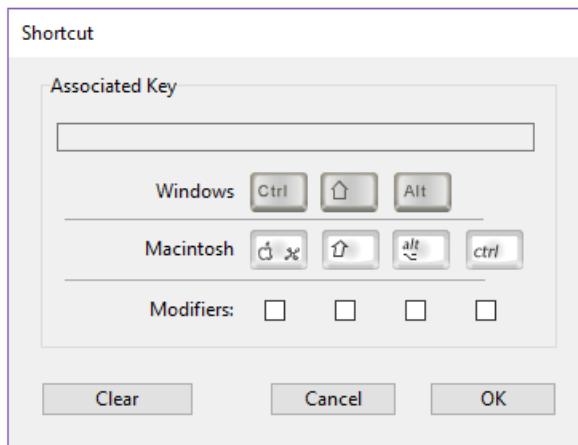
対象オブジェクト

[4D Write Pro エリア - 入力](#)

ショートカット

ボタン、ラジオボタン および チェックボックス にキーボードショートカットを割り当てることができます。ショートカットによって、ユーザーはマウスを使用しなくてもキーボードからこれらのコントロールを操作することができます。

ショートカットの設定をおこなうには、プロパティリストのショートカットプロパティの [...] ボタンをクリックします:



カスタムメニュー命令にもショートカットを割り当てることができます。2つのショートカットに衝突がある場合には、アクティブオブジェクトが優先されます。メニューへのショートカットの割り当てについては [メニュープロパティを設定する](#) を参照してください。

4D のデザイン環境で使用できるショートカットの一覧は、環境設定ダイアログの [ショートカットページ](#) にて確認できます。

JSON 文法

名称	データタイプ	とりうる値
shortcutAccel	boolean	true, false (Windows: Ctrl/macOS: Command)
shortcutAlt	boolean	true, false
shortcutCommand	boolean	true, false
shortcutControl	boolean	true, false (macOS: Control)
shortcutShift	boolean	true, false
shortcutKey	string	<ul style="list-style-type: none"> ● 任意の文字キー: "a", "b"... ● [F1]" -> "[F15]", "[Return]", "[Enter]", "[Backspace]", "[Tab]", "[Esc]", "[Del]", "[Home]", "[End]", "[Help]", "[Page up]", "[Page down]", "[left arrow]", "[right arrow]", "[up arrow]", "[down arrow]"

対象オブジェクト

[ボタン](#) - [チェックボックス](#) - [ピクチャーボタン](#) - [ラジオボタン](#)

シングルクリック編集

リストボックスにおいて、編集モードへの直接移行を可能にします。

このオプションがチェックされている場合、そのリストボックスの当該エリアが事前に選択されていたかどうかに関わらず、ユーザーのワンクリックだけでリストボックスセルを編集モードへと移行させることができます。このオプションは、リストボックスの [選択モード](#) が "なし" に設定されている場合でもセルの編集を可能にすると言う点に注意してください。

このオプションがチェックされていない場合、セルの内容を編集するにはユーザーはまず最初に編集したいセルの行を選択し、その次に編集するセルを選択する必要があります。

JSON 文法

名称	データタイプ	とりうる値
singleClickEdit	boolean	true, false

対象オブジェクト

[リストボックス](#)

フッター

フッターを表示

このプロパティは、[リストボックス列フッター](#) の表示/非表示を指定します。列ごとに 1つのフッターを表示できます。それぞれのフッターは個別に設定できます。

JSON 文法

名称	データタイプ	とりうる値
showFooters	boolean	true, false

対象オブジェクト

リストボックス

高さ

このプロパティは、リストボックスフッターの高さを 行 または ピクセル 単位で指定します。同じリストボックス内で異なる単位を使用することもできます：

- ピクセル - 指定された値は当該行に対し直接適用され、列が使用しているフォントサイズ等は考慮されません。フォントが行の高さに対して大きい場合、テキストは切り取られます。ピクチャーはフォーマットに基づき、切り取られるかリサイズされます。
- 行 - 高さは行のフォントサイズに合わせて計算されます。
 - 複数の異なるサイズが設定されている場合、4D はもっとも大きなものを使用します。たとえば、行に "Verdana 18", "Geneva 12" そして "Arial 9" が設定されている場合、4D は行の高さの決定に "Verdana 18" を使用します。複数行の場合はこの高さの倍数が使用されます。
 - この計算にはピクチャーのサイズや、フォントに適用されるスタイルは考慮されません。
 - macOS 環境下では、選択されたフォントで使用できない文字をユーザーが入力した場合、行の高さが正しくなくなる可能性があります。この場合には代理フォントが使用され、その結果サイズにはらつきが出る可能性があります。

フッターの高さは [LISTBOX SET FOOTERS HEIGHT](#) コマンドを使用して設定することもできます。

単位の変換：単位を変更した場合、4D は自動で値を再計算し、結果をプロパティリストに表示します。たとえば、使用されるフォントが "Lucida grande 24" で高さが "1 行" に設定されていれば "30 ピクセル" に、高さが "60 ピクセル" なら "2 行" になります。

単位の変更を繰り返すと、4D が自動で計算を行うため、最初の値とは結果が異なってしまうこともあります。たとえば以下のようにになります：

(Arial 18): 52 ピクセル -> 2 行 -> 40 ピクセル (Arial 12): 3 ピクセル -> 0.4 行が 1 行に切り上げられる -> 19 ピクセル

JSON 例：

```
"List Box": {  
    "type": "listbox",  
    "showFooters": true,  
    "footerHeight": "44px",  
    ...  
}
```

JSON 文法

名称	データタイプ	とりうる値
footerHeight	string	正の10進数 + px em

対象オブジェクト

[リストボックス](#)

参照

[ヘッダー - リストボックスフッター](#)

グリッド線

横線カラー

リストボックス内の横線の色を指定します (デフォルトはグレー)。

JSON 文法

名称	データタイプ	とりうる値
horizontalLineStroke	color	任意の css 値; "transparent"; "automatic"

対象オブジェクト

[リストボックス](#)

縦線カラー

リストボックス内の縦線の色を指定します (デフォルトはグレー)。

JSON 文法

名称	データタイプ	とりうる値
verticalLineStroke	color	任意の css 値; "transparent"; "automatic"

対象オブジェクト

[リストボックス](#)

ヘッダー

ヘッダーを表示

このプロパティは、[リストボックス列ヘッダー](#) の表示/非表示を指定します。列ごとに 1つのヘッダーを表示できます。それぞれのヘッダーは個別に設定できます。

JSON 文法

名称	データタイプ	とりうる値
showHeaders	boolean	true, false

対象オブジェクト

[リストボックス](#)

高さ

このプロパティは、リストボックスヘッダーの高さを 行 または ピクセル 単位で指定します。同じリストボックス内で異なる単位を使用することもできます：

- ピクセル - 指定された値は当該行に対し直接適用され、列が使用しているフォントサイズ等は考慮されません。フォントが行の高さに対して大きい場合、テキストは切り取られます。ピクチャーはフォーマットに基づき、切り取られるかリサイズされます。
- 行 - 高さは行のフォントサイズに合わせて計算されます。
 - 複数の異なるサイズが設定されている場合、4D はもっとも大きなものを使用します。たとえば、行に "Verdana 18", "Geneva 12" そして "Arial 9" が設定されている場合、4D は行の高さの決定に "Verdana 18" を使用します。複数行の場合はこの高さの倍数が使用されます。
 - この計算にはピクチャーのサイズや、フォントに適用されるスタイルは考慮されません。
 - macOS 環境下では、選択されたフォントで使用できない文字をユーザーが入力した場合、行の高さが正しくなくなる可能性があります。この場合には代理フォントが使用され、その結果サイズにはらつきが出る可能性があります。

ヘッダーの高さは [LISTBOX SET HEADERS HEIGHT](#) コマンドを使用して設定することもできます。

単位の変換：単位を変更した場合、4D は自動で値を再計算し、結果をプロパティリストに表示します。たとえば、使用されるフォントが "Lucida grande 24" で高さが "1 行" に設定されていれば "30 ピクセル" に、高さが "60 ピクセル" なら "2 行" になります。

単位の変更を繰り返すと、4D が自動で計算を行うため、最初の値とは結果が異なってしまうこともあります。たとえば以下のようにになります：

(Arial 18): 52 ピクセル -> 2 行 -> 40 ピクセル (Arial 12): 3 ピクセル -> 0.4 行が 1 行に切り上げられる -> 19 ピクセル

JSON 例：

```
"List Box": {  
    "type": "listbox",  
    "showHeaders": true,  
    "headerHeight": "22px",  
    ...  
}
```

JSON 文法

名称	データタイプ	とりうる値
headerHeight	string	正の10進数 + px em)

対象オブジェクト

[リストボックス](#)

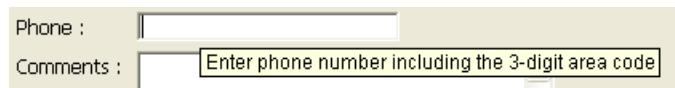
参照

[フッター - リストボックスヘッダー](#)

ヘルプ

ヘルプTips

このプロパティで、フォーム上のアクティブオブジェクトにヘルプTip を付加することができます。ランタイムにおいてマウスがオブジェクト上にあるとき、ヘルプメッセージが Tip として表示されます:



- **SET DATABASE PARAMETER** コマンドの `Tips delay` および `Tips duration` セレクターを使用することで、Tips の表示遅延や最大表示時間を指定することができます。
- **SET DATABASE PARAMETER** コマンドの `Tips enabled` セレクターを使用することで、アプリケーション全体に対してヘルプTipsを有効化あるいは無効化することができます。

オブジェクトにヘルプメッセージを関連付けるには:

- 4D の **ヘルプTipエディター** であらかじめ作成したヘルプTip を指定します。
- または、プロパティリストに直接 Tip を文字列として入力します。この方法では、XLIFF アーキテクチャーを利用することができます。XLIFF参照を指定することで、アプリケーションの言語に応じたメッセージを表示させることができます (XLIFF についての詳細は [付録 B: XLIFFアーキテクチャーを参照ください](#))。また、4D 参照を使用することもできます ([スタティックテキスト中で参照を使用する](#) 参照)。

macOSにおいては、Pop up window (32) 型のウィンドウはヘルプTips の表示ができません。

JSON 文法

名称	データタイプ	とりうる値
tooltip	テキスト	ユーザー用のヘルプ情報

対象オブジェクト

[ボタン](#) - [ボタングリッド](#) - [チェックボックス](#) - [ドロップダウンリスト](#) - [コンボボックス](#) - [階層リスト](#) - [リストボックスヘッダー](#) - [リストボックスフッター](#) - [ピクチャーボタン](#) - [ピクチャーポップアップメニュー](#) - [ラジオボタン](#)

追加のヘルプ機能

オブジェクトにヘルプTip を関連付ける方法は他にも 2通りあります:

- データベースストラクチャーレベルにおいて設定することができます (フィールドのみ)。この場合、当該フィールドが表示されるすべてのフォームにおいて、このヘルプTip が表示されます。詳細については [フィールドプロパティ](#) のヘルプTip の章を参照してください)。
- **OBJECT SET HELP TIP** コマンドを使って、カレントプロセス内で動的に設定します。

同じオブジェクトに対して複数の Tip が関連づけられている場合には、次の優先順位に従って表示されます:

1. ストラクチャーレベル (最低優先度)
2. フォームエディターレベル
3. **OBJECT SET HELP TIP** コマンド (最高優先度)

参照

[プレースホルダー](#)

階層

階層リストボックス

配列型リストボックス

このプロパティを使用してリストボックスの階層表示を設定します。JSON フォームにおいては、リストボックス列の *dataSource* プロパティの値が配列名のコレクションであるとき に階層化します。

階層リストボックス プロパティが選択されると、追加プロパティである Variable 1...10 が利用可能になります。これらには階層の各レベルとして使用するデータソース配列を指定します。これが *dataSource* の値である配列名のコレクションとなります。入力欄に値が入力されると、新しい入力欄が追加されます。10個までの変数を指定できます。これらの変数は先頭列に表示される階層のレベルを設定します。

[階層リストボックス 参照](#)

JSON 文法

名称	データタイプ	とりうる値
datasource	文字列のコレクション	階層を定義する配列名のコレクション

対象オブジェクト

[リストボックス](#)

リストボックス

列

リストボックス列のコレクション。

JSON 文法

名称	データタイプ	とりうる値
columns	列オブジェクトのコレクション。	リストボックス列のプロパティを格納します。

列オブジェクトに関してサポートされているプロパティの一覧については [列特有のプロパティ](#) の章を参照してください。

対象オブジェクト

[リストボックス](#)

詳細フォーム名

セレクション型リストボックス

リストボックスの個々のレコードを編集・表示する際に使用するフォームを指定します。

指定されたフォームは以下のタイミングで表示されます:

- リストボックスに関連付けられている `addSubrecord` (サブレコード追加)、または `editSubrecord` (サブレコード編集) の標準アクションを使用したとき ([標準アクションの使用](#) を参照してください)。
- [行をダブルクリック](#) プロパティが「レコード編集」か「レコード表示」に設定されている場合に行をダブルクリックしたとき。

JSON 文法

名称	データタイプ	とりうる値
detailForm	string	<ul style="list-style-type: none">テーブルまたはプロジェクトフォームの名前 (文字列)フォームを定義する .json ファイルへの POSIX パス (文字列)フォームを定義するオブジェクト

対象オブジェクト

[リストボックス](#)

行をダブルクリック

セレクション型リストボックス

ユーザーがリストボックスの行をダブルクリックした際に実行されるアクションを指定します。選択可能なオプションは以下の通りです:

- 何もしない (デフォルト): 行をダブルクリックしても自動アクションは発動しません。
- レコード編集: 行をダブルクリックすると、リストボックスに設定された [詳細フォーム](#) に当該レコードが表示されます レコードは読み書き可能モードで開かれるので、編集が可能です。
- レコード表示: レコード編集と同様の挙動をしますが、レコードは読み取り専用モードで開かれるため、編集はできません。

空の行へのダブルクリックは無視されます。

選択されているアクションに関わらず、`On Double Clicked` フォームイベントが生成されます。

「レコード編集」「レコード表示」のアクションに関しては `On Open Detail` フォームイベントも生成されます。リストボックスに関連付けられた詳細フォームに表示されたレコードが閉じられる際には `On Close Detail` フォームイベントが生成されます（レコードが編集されたかどうかは問いません）。

JSON 文法

名称	データタイプ	とりうる値
doubleClickInRowAction	string	"editSubrecord", "displaySubrecord"

対象オブジェクト

[リストボックス](#)

ハイライトセット

セレクション型リストボックス

このプロパティを使用して、リストボックス中でハイライトされたレコードを管理するために使用するセット名を指定します（配列 データソースが指定されている場合には、リストボックスに割り当てた変数と同じ名前の布尔配列をこの用途で使用します）。

4D は `ListBoxSetN` (N は 0 から始まり、フォーム上のリストボックスオブジェクトの数に従い増分されます) という名前のデフォルトセットを作成しますが、必要に応じてこの名前を変更できます。セットはローカル、プロセスおよびインタープロセスセットを使用できます（ネットワークトラフィックを制限するため、`$LBSet` のようなローカルセットの使用を推奨します）。指定されたセットは 4D が自動で管理します。ユーザーが 1つ以上の行を選択すると、セットは即座に更新されます。プログラムを使用して行を選択したい場合、"セット" テーマのコマンドをこのセットに適用できます。

- リストボックス行のハイライトステータスとテーブルレコードのハイライトステータスは完全に独立しています。
- "ハイライトセット" プロパティに名前が指定されていない場合、リストボックス中で行を選択することはできません。

JSON 文法

名称	データタイプ	とりうる値
highlightSet	string	セットの名称

対象オブジェクト

[リストボックス](#)

スクロールしない列とドラッグしない列

リストボックスのスクロールしない列とドラッグしない列はそれぞれ独立して動作します：

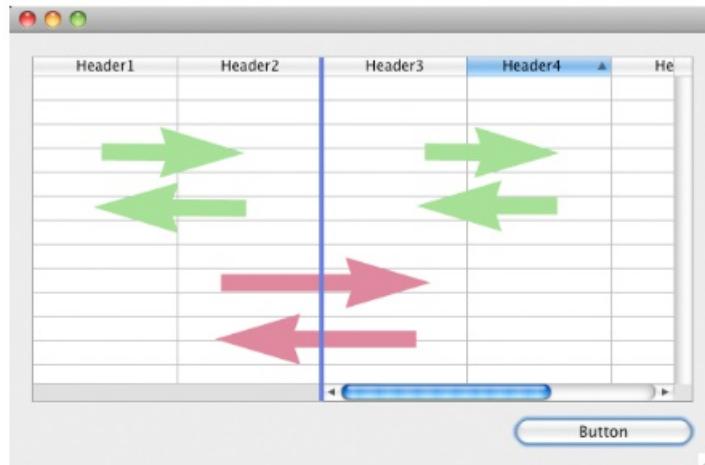
- スクロールしない列は常にリストボックスの左側に表示され、横スクロールされません。
- ドラッグしない列は、リストボックス内でドラッグ & ドロップによる列の移動ができません。

これらのプロパティはプログラミングによって設定することも可能です。詳細は [4Dランゲージリファレンス](#) マニュアルの [リストボックス](#) を参照ください。

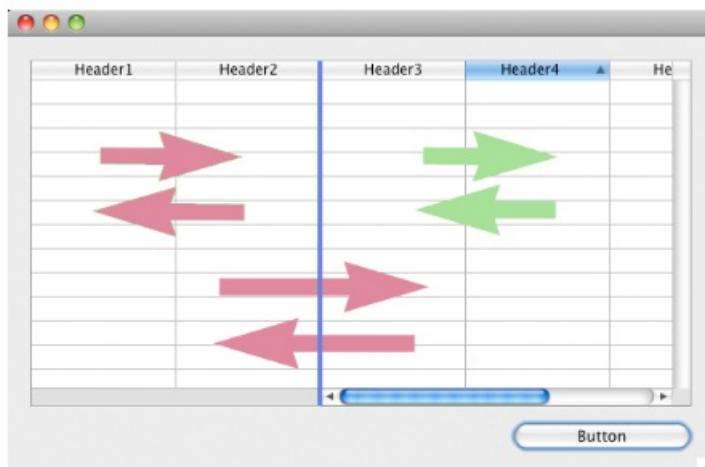
これらのプロパティは以下のように相互作用します：

- 列を "ドラッグしない" のみを設定した場合、その列は移動することができません。
- 列を "スクロールしない" のみに設定した場合、横スクロールしないエリア内に限りドラッグで列を移動することができます。しかし、そのスクロールしな

いエリアを越えて移動することはできません。



- "スクロールしない" 列と "ドラッグしない" 列を同じ数に設定した場合、スクロールしないエリア内ではドラッグで移動することもできません。



- 必要に応じてスクロールしない列数とドラッグしない列数をそれぞれ設定できます。たとえば、スクロールしない列を 3、ドラッグしない列を 1に設定した場合、ユーザーは横スクロールしないエリア内で右側 2つの列を入れ替えることができます。

スクロールしない列数

ユーザーが横スクロールしても、リストボックスの左側に常に表示される列の数を指定します。

JSON 文法

名称	データタイプ	とりうる値
lockedColumnCount	integer	最小値: 0

対象オブジェクト

リストボックス

ドラッグしない列数

実行時にドラッグで移動できない列の数を指定します。

JSON 文法

名称	データタイプ	とりうる値
staticColumnCount	integer	最小値: 0

対象オブジェクト

列数

リストボックスに表示される列の数を指定します。

[LISTBOX INSERT COLUMN](#) や [LISTBOX DELETE COLUMN](#) などのコマンドを使うことで、プログラミングによって列数を動的に変更（列の追加・削除）することができます。

JSON 文法

名称	データタイプ	とりうる値
columnCount	integer	最小値: 1

対象オブジェクト

リストボックス

行コントロール配列

配列型リストボックス

リストボックス行の表示を管理するための 4D配列です。

配列型リストボックスの任意行の "非表示"、"無効化"、"選択可能" プロパティを管理するために、この配列を使用します。このプロパティは [LISTBOX SET ARRAY](#) コマンドを使用して設定することができます。

行コントロール配列は、リストボックス内の要素数と同じ数を含んでいる倍長整数型の配列でなければなりません。行コントロール配列 の各要素は対応する行のインターフェースステータスを定義します。"リストボックス" 定数テーマの定数を使って、3つのインターフェースプロパティが利用可能です：

定数	値	説明
lk row is disabled	2	対応する行は無効化されています。テキストや、チェックボックスなどのコントロール類は暗くなっているかグレーアウトされています。入力可能なテキスト入力エリアは入力可能ではありません。デフォルト値: 有効化
lk row is hidden	1	対応する行は非表示です。行を非表示にすることは、リストボックスの見た目にのみ影響します。非表示の行は配列内には存在し、プログラミングを通して管理可能です。ランゲージコマンド（具体的には LISTBOX Get number of rows または LISTBOX GET CELL POSITION ）は行の表示/非表示のステータスを考慮しません。たとえば、10行あるリストボックスの、最初の 9行が非表示になっていた場合、 LISTBOX Get number of rows は10を返します。ユーザーの視点では、リストボックス内の非表示行の存在は視覚的には認識できません。表示されている行のみが（たとえば、"すべてを選択" コマンドなどで）選択可能です。デフォルト値: 表示
lk row is not selectable	4	対応する行は選択可能になっていません（ハイライトできません）。入力可能なテキスト入力エリアは シングルクリック編集 オプションが有効になつてない限り入力可能ではありません。しかしながら、チェックボックスなどのコントロールリストは機能しています。この設定はリストボックスの選択モードが "なし" の場合には無視されます。デフォルト値: 選択可能

行のステータスを変えるためには、対応する配列の要素に適切な定数を設定するだけです。たとえば、10行目を選択不可能に設定したい場合、以下のように書くことができます：

```
aLControlArr{10}:=lk row is not selectable
```

RowNum	Countries	Population	Landlocked
1	Luxembourg	0 502 202	<input checked="" type="checkbox"/>
2	Latvia	1 973 700	<input type="checkbox"/>
3	Kuwait	4 044 500	<input type="checkbox"/>
4	Croatia	4 284 889	<input type="checkbox"/>
5	Denmark	5 699 220	<input type="checkbox"/>
6	Nicaragua	6 071 045	<input type="checkbox"/>
7	Czech Republic	10 674 947	<input checked="" type="checkbox"/>
8	Serbia	7 306 677	<input checked="" type="checkbox"/>
9	Honduras	8 249 574	<input type="checkbox"/>
10	Austria	8 572 895	<input checked="" type="checkbox"/>
11	Hungary	10 005 000	<input checked="" type="checkbox"/>
12	Greece	10 815 197	<input type="checkbox"/>
13	Benin	10 879 829	<input type="checkbox"/>
14	Peru	10 897 770	<input type="checkbox"/>

複数のインターフェースプロパティを同時に定義することもできます:

```
aLControlArr{8}:=lk row is not selectable + lk row is disabled
```

RowNum	Countries	Population	Landlocked
1	Luxembourg	0 502 202	<input checked="" type="checkbox"/>
2	Latvia	1 973 700	<input type="checkbox"/>
3	Kuwait	4 044 500	<input type="checkbox"/>
4	Croatia	4 284 889	<input type="checkbox"/>
5	Denmark	5 699 220	<input type="checkbox"/>
6	Nicaragua	6 071 045	<input type="checkbox"/>
7	Czech Republic	10 674 947	<input checked="" type="checkbox"/>
8	Serbia	7 306 677	<input checked="" type="checkbox"/>
9	Honduras	8 249 574	<input type="checkbox"/>
10	Austria	8 572 895	<input checked="" type="checkbox"/>
11	Hungary	10 005 000	<input checked="" type="checkbox"/>
12	Greece	10 815 197	<input type="checkbox"/>
13	Benin	10 879 829	<input type="checkbox"/>
14	Peru	10 897 770	<input type="checkbox"/>

要素に対してプロパティを設定すると、(再設定しない限り) 同要素の他の値を上書きするという点に注意してください。たとえば:

```
aLControlArr{6}:=lk row is disabled + lk row is not selectable
// 6行目を無効化し、かつ選択不可に設定します
aLControlArr{6}:=lk row is disabled
// 6行目を無効化するが、選択不可を再設定していないので選択が可能となります
```

JSON 文法

名称	データタイプ	とりうる値
rowControlSource	string	行コントロール配列の名称

対象オブジェクト

リストボックス

選択モード

リストボックス行の選択モードを指定します:

- なし: 行を選択することはできません。 **シングルクリック編集** オプションがチェックされている場合を除き、リストボックスをクリックしても効果はありません。ナビゲーションキーを使用しても、リストをスクロールするだけとなり、その際に `On Selection Change` フォームイベントは生成されません。
- 単一: 一度に一行のみ選択できます。クリックすることで、行を選択できます。 `Ctrl+クリック (Windows)` や `Command+クリック (macOS)` を使うと、対象行の選択状態 (選択・非選択) が切り替わります。

上下キーを使うとリストの前後の行が選択されます。その他のナビゲーションキーはリストをスクロールします。カレントの行が変更されるたびに、`On Selection Change` フォームイベントが生成されます。

- 複数: 標準のショートカットを使用して複数行を同時に選択できます。

JSON 文法

名称	データタイプ	とりうる値
selectionMode	string	"multiple", "single", "none"

対象オブジェクト

[リストボックス](#)

オブジェクト

タイプ

必須設定です。

このプロパティは [アクティブまたはスタティックなフォームオブジェクト](#) のタイプを指定します。

JSON 文法

名称	データ タイプ	とりうる値
type	string	"button", "buttonGrid", "checkbox", "combo", "dropdown", "groupBox", "input", "line", "list", "listbox", "oval", "picture", "pictureButton", "picturePopup", "plugin", "progress", "radio", "rectangle", "ruler", "spinner", "splitter", "stepper", "subform", "tab", "text", "view", "webArea", "write"

対象オブジェクト

[4D View Pro エリア](#) - [4D Write Pro エリア](#) - ボタン - ボタングリッド - チェックボックス - コンボボックス - ドロップダウンリスト - グループボックス - 階層リスト - リストボックス - リストボックス列 - リストボックスフッター - リストボックスヘッダー - ピクチャーボタン - ピクチャーポップアップメニュー - プラグインエリア - 進捗インジケーター - ラジオボタン - スピナー - スプリッター - スタティックピクチャー - ステッパー - サブフォーム - タブコントロール - テキストエリア - [Web エリア](#)

オブジェクト名

各アクティブフォームオブジェクトにはオブジェクト名が関連付けられています。各オブジェクト名はユニークでなければなりません。

オブジェクト名のサイズ上限は 255バイトです。

4Dランゲージを使用する際、オブジェクト名を使用してアクティブフォームオブジェクトを参照できます（詳細については 4Dランゲージリファレンスの [オブジェクトプロパティ](#) を参照ください）。

フォームオブジェクトの命名規則については [識別子](#) の章を参照してください。

JSON 文法

名称	データタイプ	とりうる値
name	string	既存オブジェクトによって使用されていない、命名規則に沿った名称

対象オブジェクト

[4D View Pro エリア](#) - [4D Write Pro エリア](#) - ボタン - ボタングリッド - チェックボックス - コンボボックス - ドロップダウンリスト - グループボックス - 階層リスト - リストボックス - リストボックス列 - リストボックスフッター - リストボックスヘッダー - ピクチャーボタン - ピクチャーポップアップメニュー - プラグインエリア - 進捗インジケーター - スピナー - スプリッター - スタティックピクチャー - ステッパー - ラジオボタン - サブフォーム - タブコントロール - テキストエリア - [Web エリア](#)

値を記憶

このオプションは、フォームの [配置を記憶](#) プロパティがチェックされている場合に利用可能です。

この機能は、フォームの全体的な配置に関するオブジェクトに対してのみサポートされています。たとえばチェックボックスは、ウィンドウ内の追加エリアを表示/非表示するのに使用することができるため、このオプションが存在します。

「値」を保存できるオブジェクトは以下のとおりです：

オブジェクト	保存される値
チェックボックス	関連付けられた変数の値 (0, 1, 2)
ドロップダウンリスト	選択されている項目の番号
ラジオボタン	関連付けられた変数の値 (1, 0、ボタンにおいては true/false など型による)
タブコントロール	選択されているタブの番号

JSON 文法

名称	データタイプ	とりうる値
memorizeValue	boolean	true, false

対象オブジェクト

チェックボックス - ドロップダウンリスト - ラジオボタン - タブコントロール

変数あるいは式

セレクションおよびコレクション型のリストボックス列に関しては [式](#) の章も参照ください。

このプロパティは、データのソースを指定します。各アクティブフォームオブジェクトにはオブジェクト名と変数名が関連付けられています。変数名とオブジェクト名は違っていてもかまいません。同じフォーム内で複数のアクティブオブジェクトに同じ変数名を割り当てることができます。オブジェクト名はそれぞれユニークでなければなりません。

変数名の上限は 31バイトです。命名規則については [識別子](#) の章を参照してください。

フォームオブジェクト変数を使って、オブジェクトを監視・コントロールすることができます。たとえば、ボタンがクリックされると、その変数の値は 1 に設定されます。それ以外の場合は 0 です。進捗インジケーターに関連づけられた式は、現設定の取得・変更を可能にします。

代入可・代入不可の変数および式が使用でき、取得できるデータ型はテキスト、整数、数値、日付、時間、ピクチャー、布尔、そしてオブジェクトです。

JSON 文法

名称	データタイプ	とりうる値
dataSource	文字列、または文字列のコレクション	<ul style="list-style-type: none">4D変数、フィールド名、あるいは任意の式ダイナミック変数 の場合は、空の文字列階層リストボックス 列の場合に、文字列 (配列名) のコレクション

式

オブジェクトのデータソースとして、式 を使用することができます。有効な 4D式であれば何でも受け入れられます。シンプルな式、オブジェクトプロパティ、フォーミュラ、4D関数、プロジェクトメソッド名、標準の [Table] Field シンタックスを使用したフィールド名を使用できます。式はフォームが実行されたときに評価され、フォームイベント毎に再評価されます。式には、代入可および代入不可の式 があることに注意が必要です。

入力された式が、変数名とメソッド名の両方で使用されている場合、4Dはメソッド名が指定されたものと判断します。

ダイナミック変数

ボタン、入力オブジェクト、チェックボックス等のフォームオブジェクトに割り当てられる変数を、必要に応じて動的に、4Dに作成させることができます。これをおこなうには、"変あるいは式" プロパティを空にします (あるいは JSON の `dataSource` フィールド):

変数名が与えられていない場合、4D はフォームがロードされたときにインタークリーのプロセス変数の空間内でユニークな名前を計算し、その名前でオブジェクト用の変数を新規作成します (このメカニズムはコンパイルモードでも使用することができます)。この一時的な変数はフォームが閉じられるときに破棄されます。この方式をコンパイルモードで動作させるためには、ダイナミック変数の型を明示的に指定しなければなりません。これをおこなうには 2つの方法があります:

- プロパティリストの [式タイプ](#) を使用して型を指定する。
- たとえば `VARIABLE TO VARIABLE` コマンドを使用する、専用の初期化コードをフォームロード時に実行する。

```
If(Form event code=On Load)
    var $init : Text
    $Ptr_object:=OBJECT Get pointer(0bject named;"comments")
    $init:=""
    VARIABLE TO VARIABLE(Current process;$Ptr_object->:$init)
End if
```

4Dコード中では、`OBJECT Get pointer` コマンドで取得できるポインターを介してダイナミック変数にアクセスできます。たとえば:

```
// "tstart" オブジェクトの変数に時刻 12:00:00 を代入します
$p :=OBJECT Get pointer(0bject named;"tstart")
$p->:=?12:00:00?
```

このメカニズムを使用する利点は 2つあります:

- ひとつのホストフォーム上で複数個配置すること可能な "サブフォーム" タイプのコンポーネント開発を可能にします。たとえば、開始日と終了日を設定する 2つの日付ピッカーサブフォームをホストフォーム上に配置するケースを考えてみましょう。このサブフォームでは、日付を選択するためのオブジェクトが使用されます。開始日と終了日をそれぞれ選択できるよう、これらオブジェクトにはそれぞれ別の変数が割り当てられている必要があります。4Dにダイナミック変数を生成させることでユニークな変数を得ることができ、この問題を解決できます。
- また、メモリの利用を減少させることができます。フォームオブジェクトでは、プロセス変数とインタークロセス変数しか使用できません。しかしコンパイルモードでは、各プロセス変数のインスタンスが (サーバープロセスを含め) すべてのプロセスに対して作成されます。このインスタンスは、セッション中にフォームが使用されない場合でもメモリを消費します。フォームのロード時、4Dにダイナミック変数を作成させることで、メモリを節約できます。

配列リストボックス

配列型リストボックスの場合、リストボックスおよびリストボックス各列の変数あるいは式 プロパティには、それぞれに関連付ける配列変数の名前を指定します。ただし、リストボックスの JSON 定義においては、列の `dataSource` 値として、配列名 (文字列) のコレクションを指定すると [階層リストボックス](#) が定義されます。

対象オブジェクト

[4D View Pro エリア](#) - [4D Write Pro エリア](#) - ボタン - ボタングリッド - チェックボックス - コンボボックス - ドロップダウンリスト - 階層リスト - リストボックス - リストボックス列 - リストボックスヘッダー - リストボックスフッター - ピクチャーポップアップメニュー - プラグインエリア - 進歩インジケーター - ラジオボタン - スピナー - スプリッター - ステッパー - サブフォーム - タブコントロール - Web エリア

式の型/式タイプ

[セレクション型](#) および [コレクション型](#) のリストボックス列や、[オブジェクト](#) や [配列](#) に関連付けられた [ドロップダウンリスト](#) のプロパティリストでは、このプロパティは、[データタイプ](#) と呼ばれています。

オブジェクトに関連付けられた式または変数のデータ型を指定します。この設定の主な目的は、プロパティリスト内で提供されるテーマとオプションが、データの型と対応するようにするためにです。つまり、実際に変数の型そのものを決めるわけではありません。プロジェクトをコンパイルするには、[変数を宣言](#) する必要があります。

ただし、次の特定の場合には、このプロパティは型宣言の機能を持ちえます:

- **ダイナミック変数**: このプロパティを使って、ダイナミック変数の型を宣言することができます。
- **リストボックス列**: このプロパティは列データに表示フォーマットを関連づけるのに使用されます。提供されるフォーマットは変数型（配列型のリストボックス）またはデータ/フィールド型（セレクションおよびコレクション型のリストボックス）により異なります。使用できる標準の 4D フォーマットはテキスト、数値、整数、日付、時間、ピクチャー、そしてブールです。テキストの場合は専用の表示フォーマットがありません。標準フォーマットのほかに、定義したカスタムフォーマットも選択することができます。
- **ピクチャー変数**: このプロパティを使うと、インタープリタモードにおいてフォームロード前に変数を宣言することができます。フォーム上のピクチャー変数にピクチャーを表示する際には特別なメカニズムが使用されます。そのため、他の型の変数とは違って、ピクチャー変数の宣言は、フォームロード前（`On Load` フォームイベントよりも先）におこなう必要があります。このため、フォームを呼び出す前（たとえば `DIALOG` コマンドを呼び出す前）に `C_PICTURE(varName)` を実行するか、あらかじめプロパティリストの式の型にピクチャーを選択しておく必要があります。このいずれかをおこなわない場合、ピクチャー変数はピクチャーを正しく表示できません（インターペリタモードのみ）。

JSON 文法

名称	データ タイプ	とりうる値
dataSourceTypeHint	string	<ul style="list-style-type: none"> 標準のオブジェクト: "integer", "boolean", "number", "picture", "text", "date", "time", "arrayText", "arrayDate", "arrayTime", "arrayNumber", "collection", "object", "undefined" リストボックス列: "boolean", "number", "picture", "text", "date", "time") 配列/セレクション リストボックスのみ: "integer", "object"

対象オブジェクト

[チェックボックス](#) - [コンボボックス](#) - [ドロップダウンリスト](#) - [入力](#) - [リストボックス列](#) - [リストボックスフッター](#) - [プラグインエリア](#) - [進捗インジケーター](#) - [ラジオボタン](#) - [ルーラー](#) - [スピナー](#) - [ステッパー](#) - [サブフォーム](#) - [タブコントロール](#)

CSS クラス

[cssファイル](#) にてクラスセレクターとして使用される、(複数の場合は半角スペース区切りの) クラス名のリスト。

JSON 文法

名称	データタイプ	とりうる値
class	string	(複数の場合は半角スペース区切りの) クラス名の文字列

対象オブジェクト

[4D View Pro エリア](#) - [4D Write Pro エリア](#) - [ボタン](#) - [ボタングリッド](#) - [チェックボックス](#) - [コンボボックス](#) - [ドロップダウンリスト](#) - [グループボックス](#) - [階層リスト](#) - [リストボックス](#) - [ピクチャーボタン](#) - [ピクチャーポップアップメニュー](#) - [プラグインエリア](#) - [ラジオボタン](#) - [スタティックピクチャー](#) - [サブフォーム](#) - [テキストエリア](#) - [Web エリア](#)

コレクションまたはエンティティセレクション

コレクション要素あるいはエンティティを使用して、リストボックスの行の中身を定義することができます。

コレクションあるいはエンティティセレクションを返す式を入力します。一般的には、コレクションまたはエンティティセレクションを格納している変数名、コレクション要素、あるいはプロパティを入力します。

コレクションおよびエンティティセレクションは、フォームロード時にフォームから利用可能でなければなりません。コレクションの各要素、あるいはエンティティセレクションの各エンティティは、リストボックスの行へと割り当てられ、`This` コマンドを通してオブジェクトとして利用可能です：

- オブジェクトのコレクションを使用した場合、データソース式内で `This` コマンドを呼び出すことで、各プロパティ値にアクセスすることができます。例：`This.<propertyName>`。
- エンティティセレクションを使用した場合、データソース式内で `This` コマンドを呼び出すことで、各属性値へとアクセスすることができます。例：`This.<attributePath>`。

(オブジェクトでない) スカラー値のコレクションを使用した場合、データソース式内で This.value を呼び出すことで各値にアクセスすることができます。

ただし、この場合は値を編集したり、“カレントの項目” オブジェクトにアクセスすることはできません（以下参照）。

注：エンティティセレクションについての詳細については、[ORDA](#) の章を参照してください。

JSON 文法

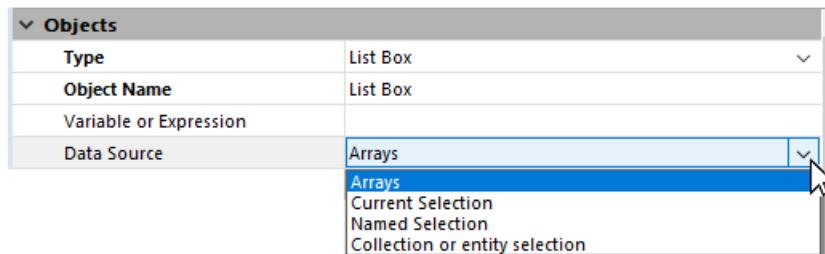
名称	データタイプ	とりうる値
dataSource	string	コレクションあるいはエンティティセレクションを返す式

対象オブジェクト

リストボックス

データソース

リストボックスの種類を指定します。



- 配列（デフォルト）：リストボックスの各行に 配列要素を割り当てます。
- カレントセレクション：指定したテーブルのカレントセレクションの各レコードごとに式、フィールド、メソッドが評価されます。
- 命名セレクション：指定した命名セレクションに含まれる各レコードごとに式、フィールド、メソッドが評価されます。
- コレクションまたはエンティティセレクション：コレクション要素あるいはエンティティを使用してリストボックスの行の中身を定義します。この場合 [コレクションまたはエンティティセレクション](#) プロパティを定義する必要があります。

JSON 文法

名称	データタイプ	とりうる値
listboxType	string	"array", "currentSelection", "namedSelection", "collection"

対象オブジェクト

リストボックス

プラグインの種類

オブジェクトに関連付ける [プラグインが提供する外部エリア](#) の名称。ここで指定する名称は、プラグインの manifest.json ファイルにて公開されています。

JSON 文法

名称	データタ イプ	とりうる値
pluginAreaKind	string	プラグインが提供する外部エリアの、% 文字で始まる名称（プロパティリストの候補表示では、この % 文字は表示されません）

対象オブジェクト

[プラグインエリア](#)

ラジオグループ

複数のラジオボタンを連動させるためのプロパティです。同じラジオグループに属している複数のラジオボタンは、一度にその内の一つのみを選択することができます。

JSON 文法

名称	データタイプ	とりうる値
radioGroup	string	ラジオグループ名

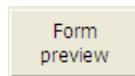
対象オブジェクト

[ラジオボタン](#)

タイトル

オブジェクトにラベルを追加します。このラベルのフォントとスタイルは指定することができます。

¥文字 (バックスラッシュまたは円マーク) を使用すると、ラベル内で強制的に改行することができます。



ラベルに ¥ を表示したい場合は ¥¥ と入力します。

デフォルトでラベルはオブジェクトの中央に置かれます。またオブジェクトにアイコンも含まれている場合、[タイトル/ピクチャー位置](#) プロパティを用いて、これら 2 つの要素の相対位置を変更することができます。

インターフェースの翻訳の目的で、XLIFF 参照をボタンのタイトルエリアに入力することができます ([付録 B: XLIFFアーキテクチャー](#) 参照)。

JSON 文法

名称	データタイプ	とりうる値
テキスト	string	なんらかのテキスト

対象オブジェクト

[ボタン](#) - [チェックボックス](#) - [リストボックスヘッダー](#) - [ラジオボタン](#) - [テキストエリア](#)

変数の計算

このプロパティは、[リストボックスフッター](#) エリアに適用される計算タイプを設定します。

リストボックスのフッターに割り当てる自動計算は [LISTBOX SET FOOTER CALCULATION](#) 4Dコマンドを使用しても設定できます。

様々な自動計算が利用可能です。以下の表は、列のデータ型に応じて使用することのできる計算と、(コードで明示的に宣言されていないとき) 4D によってフッターバリューに自動で割り当てられる型を示しています:

計算タイプ	Num	テキスト	日付	時間	Bool	ピクチャー	フッター変数の型
最小	○	○	○	○	○		列の型と同じ
最大	○	○	○	○	○		列の型と同じ
合計	○			○	○		列の型と同じ
カウント	○	○	○	○	○	○	倍長整数
平均	○			○			実数
標準偏差(*)	○			○			実数
分散(*)	○			○			実数
平方和(*)	○			○			実数
カスタム (JSON では "none")	○	○	○	○	○	○	制限なし

(*) 配列型のリストボックスのみ

フッターカルculatedには、宣言された、あるいは動的な **変数** のみを使用できます。その他の **式** (例: `Form.value`) はサポートされていません。

自動計算の際、リストボックス行の表示/非表示状態は考慮されません。表示行だけを計算対象にしたい場合、カスタムを選択してプログラムコードで計算しなくてはなりません。

Null 値は計算において無視されます。

異なる型の値がカラムに含まれる場合 (コレクションに基づいている場合など):

- 平均と合計は数値のみを計算に含めます (他の型の値は無視されます)。
- 最小と最大は、`collection.sort()` 関数にて定義されるのと同じ最小値と最大値を返します。

式に基づいている列のフッターにおいて自動計算を使用するにあたっては、次の制限があります:

- 式が "単純" なものの場合 (`[table]field` や `this.attribute` など) には、すべてのリストボックスタイプで サポートされます
- 式が "複雑" (つまり、`this.attribute` 以外) で、かつリストボックスの行数が多い場合には、コレクション/エンティティセレクションリストボックスにおいて サポートはされるものの、パフォーマンス上の理由により 推奨されません。
- 式が "複雑" な場合には、カレントセレクション/命名セレクションリストボックスにおいて サポートされません。これらの場合には、カスタム計算を使用する必要があります。

カスタム (JSON では "none") を選択した場合、4D は自動計算をおこないません。プログラムを使用して表示する値をエリアの変数に代入しなければなりません。

JSON 文法

名称	データタイプ	とりうる値
variableCalculation	string	"none", "minimum", "maximum", "sum", "count", "average", "standardDeviation", "variance", "sumSquare"

対象オブジェクト

リストボックスフッター

ピクチャー

パス名

ピクチャーボタン、ピクチャーポップアップメニュー、または [スタイルピクチャー](#) に表示させるピクチャーのパス名です。POSIX シンタックスを使用します。

ピクチャーパスに指定できる場所は次の 3箇所です:

- プロジェクトの Resources フォルダー。プロジェクト内の複数のフォームで画像を共有する場合に適切です。この場合、パス名は "/RESOURCES/<picture path>" となります。
- フォームフォルダー内の画像用フォルダー（たとえば、Images と名付けたフォルダー）。特定のフォームでしか画像が使われない場合や、そのフォームの全体を複製してプロジェクト内、または別のプロジェクトに移動させたい場合に適切です。この場合、パス名は "<picture path>" となり、フォームフォルダーを基準とした相対パスです。
- 4D のピクチャー変数。フォーム実行時に、ピクチャーがメモリに読み込まれている必要があります。この場合、パス名は "var:<variableName>"。

JSON 文法

名称	データタイプ	とりうる値
picture	text	POSIXシンタックスの相対パスまたはファイルシステムパス、ピクチャー変数の場合は "var:<variableName>"

対象オブジェクト

[ピクチャーボタン](#) - [ピクチャーポップアップメニュー](#) - [スタイルピクチャー](#)

表示

スケーリング

JSON 文法では: "scaled"

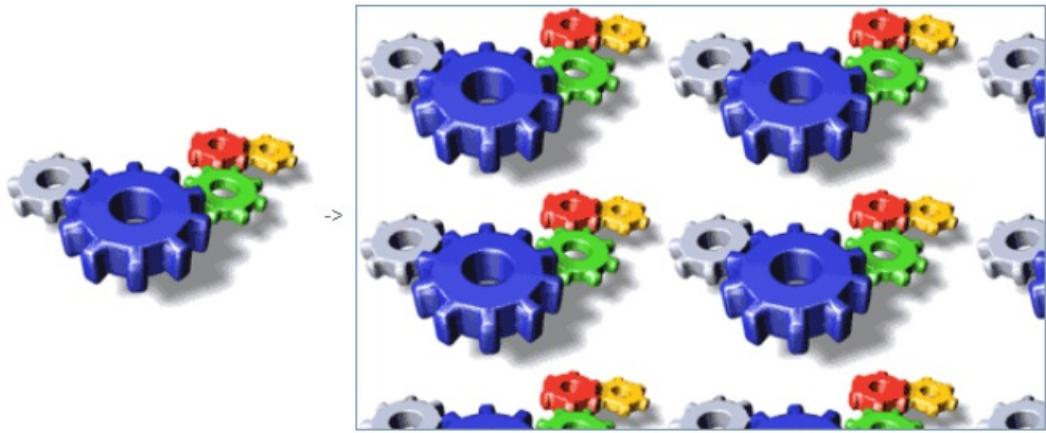
スケーリング を選択すると、ピクチャーはフィールドエリアの大きさに合うようにリサイズされます。



繰り返し

JSON 文法では: "tiled"

繰り返し フォーマットを持つピクチャーが含まれるエリアが拡大されると、ピクチャーは変形されず、エリア全体を埋めるのに必要なだけピクチャーが繰り返されます。



フィールドがオリジナルのピクチャーよりも小さいサイズにされた場合、ピクチャーはトランケート（中央合わせなし）されます。

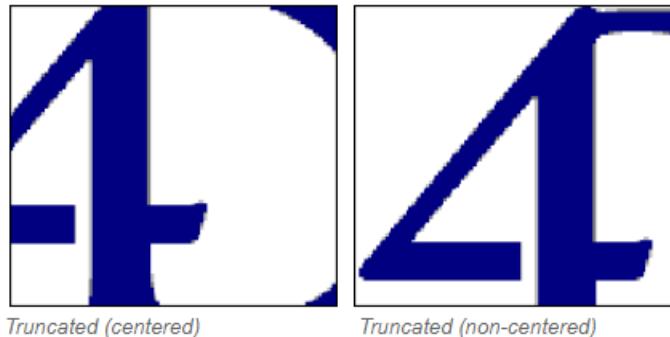
中央合わせ / トランケート（中央合わせしない）

JSON 文法では: "truncatedCenter" / "truncatedTopLeft"

中央合わせ フォーマットを選択すると、4D はエリアの中央にピクチャーを配置し、収まらない部分はエリアからはみ出します。上下、および左右のはみ出し量は同じになります。

トランケート（中央合わせしない）フォーマットを選択すると、4D はピクチャーの左上角をフィールドの左上角に合わせて配置し、フィールドエリアに収まらない部分はエリアからはみ出します。ピクチャーは右と下にはみ出します。

ピクチャーフォーマットが トランケート（中央合わせしない）の場合、入力エリアにスクロールバーを追加できます。



JSON 文法

名称	データタイプ	とりうる値
pictureFormat	string	"scaled", "tiled", "truncatedCenter", "truncatedTopLeft"

対象オブジェクト

[スタティックピクチャー](#)

プラグイン

詳細設定

プラグインの作成者が詳細オプションを提供していると、プロパティリストにて 詳細設定 ボタンが使用可能になることがあります。この場合ボタンをクリックすると、プラグインの制作元によるカスタムダイアログにてそれらのオプションを設定することができます。

この詳細設定オプションはプラグインの制作元が制御するため、詳細設定オプションに関する情報はそのプラグインの製作者から提供されます。

JSON 文法

名称	データ タイプ	とりうる値
customProperties	テキス ト	プラグイン専用のプロパティです。オブジェクトの場合は JSON 文字列として、バイナリの場合は base64エンコードの文字列としてプラグインに渡されます。

[対象オブジェクト](#)

[プラグインエリア](#)

印刷

印刷時可変

このプロパティは、レコードの中身に応じてサイズが変化しうるオブジェクトの印刷モードを管理します。これらのオブジェクト固定長フレームまたは可変長フレームでの印刷を設定することができます。固定長フレームオブジェクトは、フォーム上でオブジェクト作成するように、オブジェクトのサイズの制限内で印刷をします。可変長フレームオブジェクトはオブジェクトの中身をすべて印刷するために、印刷時に展開します。可変サイズとして印刷されるオブジェクト幅（オブジェクトプロパティによって定義）はこのオプションによって影響はされないという点に注意してください。オブジェクトの中身に応じて、高さのみが変化します。

フォーム内において複数の可変長フレームを隣同士に配置することはできません。非可変長フレームオブジェクトであれば、可変サイズで印刷されるオブジェクトのどちら側でも配置することができます。ただし、可変長フレームオブジェクトが最低でも横のオブジェクトより一行分長く、すべてのオブジェクトが上揃えで配置されていなければなりません。この条件が遵守されない場合、可変長フレームオブジェクトの水平方向の部分ごとに、ほかのフィールドのコンテンツが繰り返されます。

`Print object` と `Print form` コマンドはこのプロパティをサポートしません。

印刷オプションは次の通りです：

- 可変 オプション (印刷時可変 選択時): すべてのサブレコードが印刷されるよう、4D はフォームオブジェクトエリアを拡大あるいは縮小します。
- 固定 (切り捨て) オプション (印刷時可変 非選択時): オブジェクトエリアに表示されている内容のみを 4D は印刷します。フォームが印刷されるのは一度のみで、印刷されなかった内容は無視されます。
- 固定 (複数レコード) (サブフォームのみ): サブフォームの初期サイズを維持しますが、4D はすべてのレコードが印刷されるまで複数回にわたってフォームを印刷します。

このプロパティは `OBJECT SET PRINT VARIABLE FRAME` コマンドによって設定することができます。

JSON 文法

名称	データタイプ	とりうる値
printFrame	string	"fixed", "variable", (サブフォームのみ) "fixedMultiple"

対象オブジェクト

入力 - サブフォーム (リストサブフォームのみ) - 4D Write Pro エリア

値の範囲

デフォルト値

入力オブジェクトにデフォルト値を割り当てるすることができます。このプロパティは、入力オブジェクトに設定されている [変数あるいは式](#) がフィールドであるときに便利です。新規レコードが作成され、初めて表示されるときにデフォルト値が代入されます。エリアが [入力不可](#) に設定されていなければ、デフォルト値を書き換えることができます。

デフォルト値を指定できるのは、[式の型](#) が次のいずれかの場合です：

- テキスト/文字列
- 数値/整数
- date
- time
- boolean

日付、時刻、シーケンス番号については、4D が提供する記号を使用することができます。日付と時刻はシステムから取得されます。シーケンス番号は 4D が自動で生成します。自動で使用できるデフォルト値の記号は以下の通りです：

記号	意味
#D	本日の日付
#H	現在の時刻
#N	シーケンス番号

カレントデータファイルの特定のテーブルにおいて、レコード毎のユニーク番号を生成するためにシーケンス番号を使用することができます。シーケンス番号は倍長整数型で新規レコード毎に生成されます。番号は 1 から始まり、1ずつ増加します。シーケンス番号が割り当てられたレコードがテーブルから削除されても、その番号は再利用されません。シーケンス番号は各テーブルが保有する内部カウンターが管理します。詳細は [自動インクリメント](#) の段落を参照してください。

このプロパティと、リストボックス列に固定値を表示させるための "デフォルト値" を混同しないようにしてください。

JSON 文法

名称	データタイプ	とりうる値
defaultValue	string, number, date, time, boolean	任意の値。次の記号を含む: "#D", "#H", "#N"

対象オブジェクト

入力

除外リスト

除外リストを使用すると、当該リストの項目は入力できなくなります。ユーザーがこのリストに含まれる値を入力したとき、その入力は自動的に却下され、エラーメッセージが表示されます。

階層リストを指定した場合は、第一レベルの項目のみが考慮されます。

JSON 文法

名称	データタイプ	とりうる値
excludedList	list	除外する値のリスト

対象オブジェクト

[コンボボックス - リストボックス列 - 入力](#)

指定リスト

有効な入力値のリストを指定するために使用します。たとえば、役職名のリストを指定リストとして設定できます。こうすると、事前に作成されたリスト中の役職名だけ有効な値となります。

指定リストを指定しても、フィールドが選択されたときにリストは自動で表示されません。指定リストを表示したい場合は、"データソース"テーマの [選択リスト](#) プロパティに同じリストを指定します。ただし、[選択リスト](#) プロパティだけが指定されているときは異なり、指定リストも設定されている場合にはキーボードによる入力はできません。ポップアップメニューでのリスト項目の選択だけが可能です。[選択リスト](#) と指定リストにそれぞれ別のリストが指定されている場合、指定リストが優先されます。

階層リストを指定した場合は、第一レベルの項目のみが考慮されます。

JSON 文法

名称	データタイプ	とりうる値
requiredList	list	有効な入力値のリスト

対象オブジェクト

[コンボボックス - リストボックス列 - 入力](#)

リサイズオプション

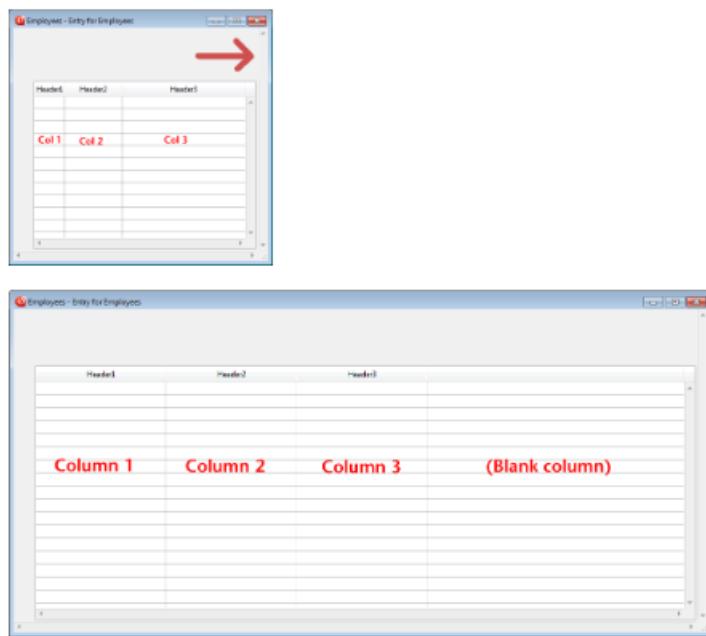
カラム自動リサイズ

このプロパティがチェックされているとき (JSON の値は `rightToLeft`)、リストボックス列は定義された **最大幅** と **最小幅** の範囲の中でリストボックスと共に自動的にリサイズされます。

このプロパティがチェックされていないときには (JSON の値は `legacy`)、リストボックス内で最も右のカラムのみが (定義された最大幅を超えたとしても) リサイズされます。

カラムの自動リサイズの仕組み

- リストボックスの幅を拡大すると、その列も一つずつ、右から左へと順に、**最大幅** に達するまで拡大されていきます。[サイズ変更可](#) プロパティがチェックされている列だけがリサイズされます
- リストボックスの幅を縮小するときも同じ手順が適用されますが、順番が逆になります (左から右へと列がリサイズされていきます)。列の幅がそれぞれ **最小幅** に達すると、水平スクロールバーがアクティブになります。
- カラムは水平スクロールバーが "アクティブ" でない場合にのみリサイズされます。つまり、現サイズでリストボックスのすべての列が完全に表示されている場合のみです。注: 水平スクロールバーの表示/非表示は、アクティブ/非アクティブとは関係ありません。スクロールバーは、非表示かつアクティブであることが可能です。
- すべての列が最大幅に到達すると、これらはそれ以上拡大されず、余分な空白を埋める形で空の列が右に追加されます。この余白カラムがあるときにリストボックスの幅を縮小させた場合、余白カラムから先に縮小されていきます。



余白カラムについて

余白カラムの見た目は既存の列と同じになります。既存のリストボックスカラムにヘッダー/フッターがある場合には余白カラムにもこれらの要素があり、同じ背景色が適用されます。

余白カラムのヘッダー/フッターはクリック可能ですが、他のカラムには何の影響も及ぼしません (つまり並び替えなどはおこなわれません)。しかしながら、`On Clicked`、`On Header Click` そして `On Footer Click` イベントはそれぞれ生成されます。

余白カラム内のセルがクリックされた場合、[LISTBOX GET CELL POSITION](#) コマンドは列番号として "X+1" を返します (X は既存の列数です)。

JSON 文法

名称	データタイプ	とりうる値
resizingMode	string	"rightToLeft", "legacy"

対象オブジェクト

[リストボックス](#)

横方向サイズ変更

このプロパティは、ユーザーがフォームの幅をサイズ変更したときの、当該オブジェクトの挙動を指定します。このプロパティは `OBJECT SET RESIZING OPTIONS` ランゲージコマンドによっても設定することができます。

次の値が提供されています:

オプション	JSON 値	戻り値
拡大	"grow"	ユーザーがウィンドウの幅を変更すると、オブジェクトの幅にも同じ割合を適用します。
移動	"move"	ユーザーがウィンドウの幅を変更すると、幅の増加分と同じだけオブジェクトを左か右に移動します。
なし	"fixed"	フォームサイズが変更されても、オブジェクトは固定されたままでです。

このプロパティは [縦方向サイズ変更](#) プロパティと連携して機能します。

JSON 文法

名称	データタイプ	とりうる値
sizingX	string	"grow", "move", "fixed"

対象オブジェクト

[4D View Pro エリア](#) - [4D Write Pro エリア](#) - [ボタン](#) - [ボタングリッド](#) - [チェックボックス](#) - [コンボボックス](#) - [ドロップダウンリスト](#) - [グループボックス](#) - [階層リスト](#) - [入力](#) - [リストボックス](#) - [線](#) - [リストボックス列](#) - [楕円](#) - [ピクチャーボタン](#) - [ピクチャーポップアップメニュー](#) - [プラグインエリア](#) - [進捗インジケーター](#) - [ラジオボタン](#) - [ルーラー](#) - [四角](#) - [スピナー](#) - [スプリッター](#) - [スタティックピクチャー](#) - [ステッパー](#) - [サブフォーム](#) - [タブコントロール](#) - [Web エリア](#)

縦方向サイズ変更

このプロパティは、ユーザーがフォームの高さをサイズ変更したときの、当該オブジェクトの挙動を指定します。このプロパティは `OBJECT SET RESIZING OPTIONS` ランゲージコマンドによっても設定することができます。

次の値が提供されています:

オプション	JSON 値	戻り値
拡大	"grow"	ユーザーがウィンドウの高さを変更すると、オブジェクトの高さにも同じ割合を適用します。
移動	"move"	ユーザーがウィンドウの高さを変更すると、高さの変更分と同じだけオブジェクトを上か下に移動します。
なし	"fixed"	フォームサイズが変更されても、オブジェクトは固定されたままでです。

このプロパティは [横方向サイズ変更](#) プロパティと連携して機能します。

JSON 文法

名称	データタイプ	とりうる値
sizingY	string	"grow", "move", "fixed"

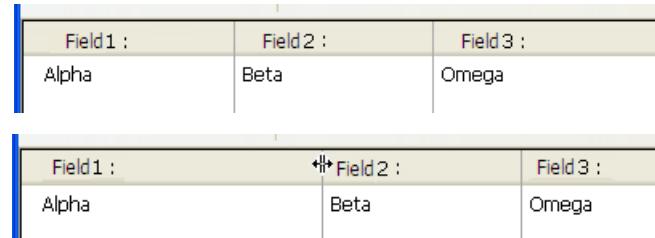
対象オブジェクト

4D View Pro エリア - 4D Write Pro エリア - ボタン - ボタングリッド - チェックボックス - コンボボックス - ドロップダウンリスト - グループボックス - 階層リスト - 入力 - リストボックス - 線 - リストボックス列 - 橋円 - ピクチャーボタン - ピクチャーポップアップメニュー - ブラグインエリア - 進捗インジケーター - ラジオボタン - ルーラー - 四角 - スピナー - スプリッター - ステイックピクチャー - ステッパー - サブフォーム - タブコントロール - Web エリア

以降のオブジェクトを移動する

プロパティを適用するとスプリッターオブジェクトは "プッシャー" になり、そのオブジェクトの右側（垂直スプリッター）または下側（水平スプリッター）にある他のオブジェクトは、スプリッターと一緒に押し出されて移動します。

"プッシャー" スプリッターを移動した場合の結果を次に示します：



スプリッターに対してこのプロパティを指定しない場合、結果は次のようにになります：



JSON 文法

名称	データタイプ	とりうる値
splitterMode	string	"move" (プッシャー), "resize" (標準)

対象オブジェクト

スプリッター

サイズ変更可

このオプションが選択されていると、ユーザーはヘッダーエリアの右側をドラッグすることで列のサイズを変更できます。

JSON 文法

名称	データタイプ	とりうる値
resizable	boolean	"true", "false"

対象オブジェクト

リストボックス列

スケール

バーバーショップ

このプロパティは、サーモメーターの "バーバーショップ" バリエーションを有効にします。

JSON 文法

名称	データタイプ	とりうる値
max	number	プロパティが存在しない = 有効; 存在する = 無効 (デフォルトサーモメーター)

対象オブジェクト

[バーバーショップ](#)

目盛りを表示

ラベルの隣に目盛を表示、または非表示にします。

JSON 文法

名称	データタイプ	とりうる値
showGraduations	boolean	"true", "false"

対象オブジェクト

[デフォルトサーモメーター - ルーラー](#)

目盛りのステップ

目盛の表示単位です。

JSON 文法

名称	データタイプ	とりうる値
graduationStep	integer	最小値: 0

対象オブジェクト

[デフォルトサーモメーター - ルーラー](#)

ラベル位置

ラベルが表示される際の位置です。

- なし - ラベルは表示されません。
- 上 - インジケーターの上または左にラベルを表示します。

- 下 - インジケーターの下または右にラベルを表示します。

JSON 文法

名称	データタイプ	とりうる値
labelsPlacement	string	"none", "top", "bottom", "left", "right"

対象オブジェクト

[デフォルトサーモメーター - ルーラー](#)

最大

インジケーターの最大値です。

- 時間型のステッパーの場合、値は秒を表します。日付型のステッパーでは、最小および最大プロパティは無視されます。
- [バーバーショップサーモメーター](#) を有効にするには、このプロパティを取り除きます。

JSON 文法

名称	データタイプ	とりうる値
max	string / number	最小値: 0 (数値型の場合)

対象オブジェクト

[デフォルトサーモメーター - ルーラー - ステッパー](#)

最小

インジケーターの最小値です。時間型のステッパーの場合、値は秒を表します。日付型のステッパーでは、最小および最大プロパティは無視されます。

JSON 文法

名称	データタイプ	とりうる値
min	string / number	最小値: 0 (数値型の場合)

対象オブジェクト

[デフォルトサーモメーター - ルーラー - ステッパー](#)

ステップ

使用時に各値の間にあけることができる最小の間隔です。時間型のステッパーの場合、このプロパティは秒を表します。日付型のステッパーでは日数を表します。

JSON 文法

名称	データタイプ	とりうる値
step	integer	最小値: 1

対象オブジェクト

デフォルトサーモメーター - ルーラー - ステッパー

サブフォーム

削除を許可

リストサブフォーム内でユーザーがサブレコードを削除できるかどうかを指定します。

JSON 文法

名称	データタイプ	とりうる値
deletableInList	boolean	true, false (デフォルトは true)

対象オブジェクト

[サブフォーム](#)

詳細フォーム

このプロパティを使用して、サブフォームで使用する詳細フォームを割り当てます。以下のものを使用できます：

- ウィジェット (ページタイプのサブフォームで、特定の機能を実現するために作成されています)。この場合、[リストフォーム](#) および [ソース](#) プロパティは存在しないか、空でなくてはいけません。
コンポーネントで公開されていれば、コンポーネントフォーム名を選べます。

サブフォームを介して追加の機能を提供する [コンポーネント](#) を作成することが可能です。

- [リストサブフォーム](#) に関する詳細フォーム。詳細フォームはサブレコードを入力したり表示したりするために使用します。通常、詳細フォームにはリストサブフォームより多くの情報が含まれています。詳細フォームは、サブフォームと同じテーブルに属していないかもしれません。典型的には、出力フォームをリストフォーム に、入力フォームを詳細フォームに指定します。詳細フォームを指定しない場合、4Dは自動でテーブルのデフォルト入力フォームを使用します。

JSON 文法

名称	データタイプ	とりうる値
detailForm	string	テーブルまたはプロジェクトフォームの名前 (文字列), フォームを定義する .json ファイルへの POSIX パス (文字列), またはフォームを定義するオブジェクト

対象オブジェクト

[サブフォーム](#)

空行をダブルクリック

リストサブフォームの空行がダブルクリックされた際に実行されるアクションを指定します。次のオプションから選択することができます：

- 何もしない: ダブルクリックを無視します。
- レコード追加: サブフォーム中に新規レコードを作成し、編集モードにします。"リスト更新可" オプションが選択されている場合、レコードは直接リスト内に作成されます。選択されていない場合、レコードはサブフォームに割り当てられた [詳細フォーム](#) 上に作成されます。

JSON 文法

名称	データタイプ	とりうる値
doubleClickInEmptyAreaAction	string	"addSubrecord", 何もしない場合は ""

対象オブジェクト

サブフォーム

参照

行をダブルクリック

行をダブルクリック

リストサブフォーム

ユーザーがリストサブフォームの行をダブルクリックした際に実行されるアクションを指定します。選択可能なオプションは以下の通りです：

- 何もしない（デフォルト）：行をダブルクリックしても自動アクションは発動しません。
- レコード編集：行をダブルクリックすると、リストサブフォームに設定された [詳細フォーム](#) に当該レコードが表示されます。レコードは読み書き可能モードで開かれるので、編集が可能です。
- レコード表示：レコード編集と同様の挙動をしますが、レコードは読み取り専用モードで開かれるため、編集はできません。

選択されているアクションに関わらず、 `On Double Clicked` フォームイベントが生成されます。

「レコード編集」「レコード表示」のアクションに関しては `On Open Detail` フォームイベントも生成されます。リストボックスに関連付けられた詳細フォームに表示されたレコードが閉じられる際には `On Close Detail` フォームイベントが生成されます（レコードが編集されたかどうかは問いません）。

JSON 文法

名称	データタイプ	とりうる値
doubleClickInRowAction	string	"editSubrecord", "displaySubrecord"

対象オブジェクト

サブフォーム

参照

空行をダブルクリック

リスト更新可

リストサブフォームでこのプロパティが有効化されると、ユーザーはリスト内で直接レコードデータを更新できます（この場合、関連づけられている [詳細フォーム](#) は開きません）。

これをおこなうには、更新するフィールド上で 2回クリックをおこない、編集モードにします（ダブルクリックにならないようクリックの間隔をあけなければなりません）。

JSON 文法

名称	データタイプ	とりうる値
enterableInList	boolean	true, false

対象オブジェクト

サブフォーム

リストフォーム

このプロパティを使用して、サブフォームで使用するリストフォームを割り当てます。リストサブフォームを使うことで、他のテーブルのデータを入力、表示、および更新することができます。

リストサブフォームをデータ入力に使用するには 2つの方法があります。一つはユーザーがサブフォームに直接データを入力する方法です。もう一つは [入力フォーム](#) を開いてデータを入力する方法です。後者の設定では、サブフォームとして使用されるフォームがリストフォーム、入力のために使用されるフォームが詳細フォームとなります。

JSON 文法

名称	データタイプ	とりうる値
listForm	string	テーブルまたはプロジェクトフォームの名前 (文字列), フォームを定義する .json ファイルへの POSIX パス (文字列), またはフォームを定義するオブジェクト

対象オブジェクト

サブフォーム

ソース

リストサブフォームが属するテーブル (あれば) を指定します。

JSON 文法

名称	データタイプ	とりうる値
table	string	4D テーブル名, テーブルなしの場合は ""

対象オブジェクト

サブフォーム

選択モード

リストボックス行の選択モードを指定します:

- なし: 行を選択することはできません。[リスト更新可](#) オプションがチェックされている場合を除き、リストをクリックしても効果はありません。ナビゲーションキーを使用しても、リストをスクロールするだけとなり、その際に `On Selection Change` フォームイベントは生成されません。
- 単一: 一度に一行のみ選択できます。クリックすることで、行を選択できます。Ctrl+クリック (Windows) や Command+クリック (macOS) を使うと、対象行の選択状態 (選択・非選択) が切り替わります。
上下キーを使うとリストの前後の行が選択されます。その他のナビゲーションキーはリストをスクロールします。カレントの行が変更されるたびに、`On Selection Change` フォームイベントが生成されます。
- 複数: 標準のショートカットを使用して複数行を同時に選択できます。
 - 選択されたサブレコードは `GET HIGHLIGHTED RECORDS` で取得できます。
 - レコードはクリックにより選択されますが、カレントレコードは変更されません。
 - Ctrl+クリック (Windows) や Command+クリック (macOS) を使うと、対象レコードの選択状態 (選択・非選択) が切り替わります。
上下キーを使うとリストの前後のレコードが選択されます。その他のナビゲーションキーはリストをスクロールします。選択レコードが変更されるたびに、`On Selection Change` フォームイベントが生成されます。

JSON 文法

名称	データタイプ	とりうる値
selectionMode	string	"multiple", "single", "none"

対象オブジェクト

[サブフォーム](#)

テキスト

ピッカーの使用を許可

このプロパティが有効化されると、[OPEN FONT PICKER](#) または [OPEN COLOR PICKER](#) を使用してユーザーがフォントピッカー/カラーピッカーを呼び出すことを許可します。これらのピッカーウィンドウを使用して、ユーザーはフォームオブジェクトのフォントやカラーをクリックによって変更できます。このプロパティが無効になっていると（デフォルト）、ピッカーを開くコマンドは使用できません。

JSON 文法

プロパティ	データタイプ	とりうる値
allowFontColorPicker	boolean	false (デフォルト), true

対象オブジェクト

[入力](#)

太字

選択テキストの線を太くし、濃く見えるようにします。

このプロパティは [OBJECT SET FONT STYLE](#) コマンドによって設定することができます。

これは通常のテキストです。

これは太字のテキストです。

JSON 文法

プロパティ	データタイプ	とりうる値
fontWeight	テキスト	"normal", "bold"

対象オブジェクト

[ボタン](#) - [チェックボックス](#) - [コンボボックス](#) - [ドロップダウンリスト](#) - [グループボックス](#) - [階層リスト](#) - [入力](#) - [リストボックス](#) - [リストボックス列](#) - [リストボックス](#) フッター - [リストボックスヘッダー](#) - [ラジオボタン](#) - [テキストエリア](#)

イタリック

選択テキストの線を右斜めに傾けます。

このプロパティは [OBJECT SET FONT STYLE](#) コマンドによって設定することができます。

これは通常のテキストです。

これはイタリックのテキストです。

JSON 文法

名称	データタイプ	とりうる値
fontStyle	string	"normal", "italic"

対象オブジェクト

ボタン - チェックボックス - コンボボックス - ドロップダウンリスト - グループボックス - 階層リスト - 入力 - リストボックス - リストボックス列 - リストボックスフッター - リストボックスヘッダー - ラジオボタン - テキストエリア

下線

選択テキストの下に線を引きます。

これは通常のテキストです。
これは下線の付いたテキストです。

JSON 文法

名称	データタイプ	とりうる値
textDecoration	string	"normal", "underline"

対象オブジェクト

ボタン - チェックボックス - コンボボックス - ドロップダウンリスト - グループボックス - 階層リスト - 入力 - リストボックス - リストボックス列 - リストボックスフッター - リストボックスヘッダー - ラジオボタン - テキストエリア

フォント

このプロパティは、オブジェクトで使用される フォントテーマ または フォントファミリー を指定します。

フォントテーマと フォントファミリー プロパティは、どちらか一方しか指定できません。フォントテーマは、サイズを含めたフォント属性を定めます。フォントファミリーの場合は、フォント名・フォントサイズ・フォントカラーをそれぞれ定義することができます。

フォントテーマ

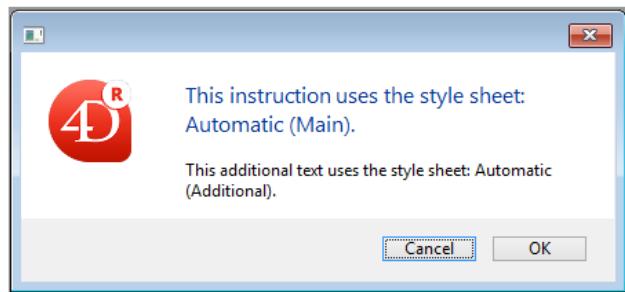
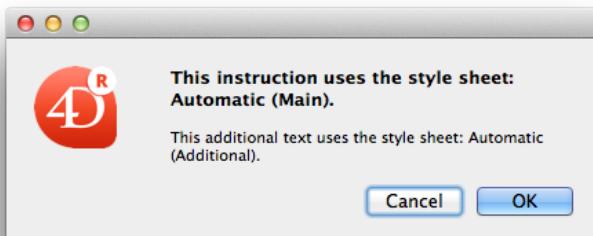
フォントテーマプロパティには、自動スタイルの名前を指定します。自動スタイルは、オブジェクトに使われるフォントファミリー・フォントサイズ・フォントカラーをシステムパラメーターに応じて動的に定めます。これらのパラメーターは次に依存します:

- プラットフォーム
- システム言語
- フォームオブジェクトのタイプ

フォントテーマを使うことで、システムの現インターフェース標準に沿うようにタイトルが表示されることが保証されます。ただし、マシンごとにサイズが変わることもかもしれません。

3つのフォントテーマが提供されています:

- normal: フォームエディター内で作成された新規オブジェクトにデフォルトで適用される自動スタイルです。
- main および additional フォントテーマは テキストエリア と 入力 オブジェクトでのみサポートされています。これらのテーマは、おもにダイアログボックスのデザインを目的に提供されています。インターフェースウィンドウにおいて main フォントテーマは本文用、additional テーマは詳細情報を追記するためのものです。下に macOS および Windows にてこれらのフォントテーマを使ったダイアログボックスの例を示します:



フォントテーマはフォントだけでなく、サイズやカラーも定めます。一部のカスタムスタイルプロパティ（太字、イタリック、下線）は動作に影響なく適用することができます。

JSON 文法

名称	データタイプ	とりうる値
fontTheme	string	"normal", "main", "additional"

対象オブジェクト

ボタン - チェックボックス - コンボボックス - ドロップダウンリスト - グループボックス - 階層リスト - 入力 - リストボックス - リストボックス列 - リストボックス
フッター - リストボックスヘッダー - ラジオボタン - テキストエリア

フォントファミリー

次の 2種類のフォントファミリーが存在します:

- フォントファミリー: "times", "courier", "arial" などのフォントファミリーの名称。
- 総称ファミリー: "serif", "sans-serif", "cursive", "fantasy", "monospace" などの汎用ファミリーの名称。

このプロパティは [OBJECT SET FONT](#) コマンドによって設定することができます。

これは 游ゴシック フォントです。

これは 游明朝 フォントです。

JSON 文法

名称	データタイプ	とりうる値
fontFamily	string	CSS フォントファミリー名

4D では Webセーフ フォントだけを使うことを推奨しています。

対象オブジェクト

ボタン - チェックボックス - コンボボックス - ドロップダウンリスト - グループボックス - 階層リスト - 入力 - リストボックス - リストボックス列 - リストボックス
フッター - リストボックスヘッダー - ラジオボタン - テキストエリア

フォントサイズ

文字の大きさをポイントで指定します。

JSON 文法

名称	データタイプ	とりうる値
fontSize	integer	フォントサイズ (ポイント単位) 最小値: 0

対象オブジェクト

ボタン - チェックボックス - コンボボックス - ドロップダウンリスト - グループボックス - 階層リスト - 入力 - リストボックス - リストボックス列 - リストボックス
フッター - リストボックスヘッダー - ラジオボタン - テキストエリア

フォントカラー

文字の色を指定します。

オブジェクトの [境界線スタイル](#) に "標準" または "ドット" を選択している場合、このプロパティはその境界線の色も指定します。

カラーは次の方法で指定できます:

- カラーネーム - 例: "red"
- 16進数値 - 例: "#ff0000"
- RGB値 - 例: "rgb(255,0,0)"

このプロパティは

[OBJECT SET RGB COLORS**](#) コマンドによって設定することができます。

JSON 文法

名称	データタイプ	とりうる値
stroke	string	任意の css 値; "transparent"; "automatic"

対象オブジェクト

ボタン - チェックボックス - コンボボックス - ドロップダウンリスト - グループボックス - 階層リスト - 入力 - リストボックス - リストボックス列 - リストボックス
フッター - リストボックスヘッダー - 進捗インジケーター - ルーラー - ラジオボタン - テキストエリア

フォントカラー式

セレクションおよびコレクション/エンティティセレクション型のリストボックス

リストボックスの各行にカスタマイズしたフォントカラーを適用するために使用します。RGBカラーを使用しなければなりません。この点に関する詳細は [4D ランゲージリファレンスマニュアル](#) の [OBJECT SET RGB COLORS](#) コマンドの説明を参照してください。

式または変数 (配列を除く) を入力します。表示される行ごとに式や変数は評価されます。ここでは [SET RGB COLORS](#) テーマの定数を使用することができます。

また、このプロパティは [LISTBOX SET PROPERTY](#) コマンドに `lk font color expression` 定数を指定して設定することもできます。

このプロパティは [メタ情報式](#) を使用しても設定することができます。

以下の例は変数名を使用しています。フォントカラー式 に `CompanyColor` を入力し、フォームメソッドに以下のコードを書きます:

```
CompanyColor:=Choose([Companies]ID;Background color;Light shadow color;  
Foreground color;Dark shadow color)
```

JSON 文法

名称	データタイプ	とりうる値
rowStrokeSource	string	フォントカラー式

対象オブジェクト

[リストボックス](#)

スタイル式

セレクションおよびコレクション/エンティティセレクション型のリストボックス

リストボックスの各行にカスタマイズされた文字スタイルを適用するために使用します。

式または変数 (配列を除く) を入力します。式や変数は、表示行ごと (リストボックスのプロパティの場合) または表示セルごと (リストボックス列のプロパティの場合) に評価されます。ここでは [Font Styles](#) テーマの定数を使用することができます。

例:

```
Choose([Companies]ID;Bold;Plain;Italic;Underline)
```

また、このプロパティは `LISTBOX SET PROPERTY` コマンドに `lk font style expression` 定数を指定して設定することもできます。

このプロパティは [メタ情報式](#) を使用しても設定することができます。

JSON 文法

名称	データタイプ	とりうる値
rowStyleSource	string	表示される行/セルごとに評価されるスタイル式。

対象オブジェクト

[リストボックス - リストボックス列](#)

横揃え

エリア中のテキストの横位置を指定します。

JSON 文法

名称	データタイプ	とりうる値
textAlign	string	"automatic", "right", "center", "justify", "left"

対象オブジェクト

[グループボックス - リストボックス - リストボックス列 - リストボックスヘッダー - リストボックスフッター - テキストエリア](#)

縦揃え

エリア中のテキストの縦位置を指定します。

デフォルト オプション (JSON値: `automatic`) の場合は、各列のデータ型に基づき整列方向が決定されます:

- ピクチャーを除き、すべて 下 です。
- ピクチャーは 上 です。

このプロパティは、[OBJECT Get vertical alignment](#) と [OBJECT SET VERTICAL ALIGNMENT](#) コマンドを使用して設定することもできます。

JSON 文法

名称	データタイプ	とりうる値
verticalAlign	string	"automatic", "top", "middle", "bottom"

対象オブジェクト

[リストボックス - リストボックス列 - リストボックスフッター - リストボックスヘッダー](#)

メタ情報式

コレクションまたはエンティティセレクション型リストボックス

表示される行ごとに評価される式あるいは変数を指定します。行テキスト属性全体を定義することができます。オブジェクト変数、あるいはオブジェクトを返す式 を指定する必要があります。以下のオブジェクトプロパティがサポートされています:

プロパティ名	タイプ	説明
stroke	string	フォントカラー。任意の CSSカラー (例: "#FF00FF"), "automatic", "transparent"
fill	string	背景色。任意の CSSカラー (例: "#F00FFF"), "automatic", "transparent"
fontStyle	string	"normal", "italic"
fontWeight	string	"normal", "bold"
textDecoration	string	"normal", "underline"
unselectable	boolean	対応する行が選択不可 (つまりハイライトすることができない状態) であることを指定します。このオプションが有効化されている場合、入力可能エリアは入力可能ではなくなります (ただし "シングルクリック編集" オプションが有効化されている場合を除く)。チェックボックスやリストといったコントロール類は引き続き稼働します。この設定はリストボックスの選択モードが "なし" の場合には無視されます。デフォルト値: false
disabled	boolean	対応する行を無効化します。このオプションが有効化されると、入力可能エリアは入力可能ではなくなります。テキストや、(チェックボックス、リストなどの) コントロール類は暗くなっているかグレーアウトされます。デフォルト値: false
cell. <columnName>	object	プロパティを单一のカラムに適用するときに使用します。<columnName> には、リストボックスカラムのオブジェクト名を渡します。注: "unselectable" および "disabled" プロパティは行レベルでのみ定義可能です。"セル" オブジェクトに指定した場合、これらは無視されます。

このプロパティで設定されたスタイルは、プロパティリスト内で他のスタイル設定が式により定義されている場合には無視されます ([スタイル式](#)、[フォントカラー式](#)、[背景色式](#))。

例題

`Color` プロジェクトメソッドには、以下のコードを書きます:

```

// Color メソッド
// 特定の行に対してフォントカラーを、そして特定のカラムに対して背景色を設定します:
C_OBJECT($0)
Form.meta:=New object
If(This.ID>5) // ID はコレクションオブジェクト/エンティティの属性です
  Form.meta.stroke:="purple"
  Form.meta.cell:=New object("Column2";New object("fill";"black"))
Else
  Form.meta.stroke:="orange"
End if
$0:=Form.meta

```

ベストプラクティス: このような場合には最適化のため、フォームメソッド内で `meta.cell` オブジェクトを作成しておくことが推奨されます。

```

// フォームメソッド
Case of
:(Form event code=On Load)
  Form.colStyle:=New object("Column2";New object("fill";"black"))
End case

```

Color メソッドには、以下のコードを書きます:

```

// Color メソッド
...
If(This.ID>5)
  Form.meta.stroke:="purple"
  Form.meta.cell:=Form.colStyle // より良いパフォーマンスのため、同じオブジェクトを再利用します
...

```

This コマンドも参照してください。

JSON 文法

名称	データタイプ	とりうる値
metaSource	string	表示される行/セルごとに評価されるオブジェクト式。

対象オブジェクト

リストボックス

マルチスタイル

このプロパティは、選択エリアでスタイルの利用を可能にするかどうかを指定するものです。プロパティリストでこのオプションがチェックされていると、4D はエリア中の `` HTML タグをスタイル属性として解釈します。

デフォルトでは、このオプションは有効化されていません。

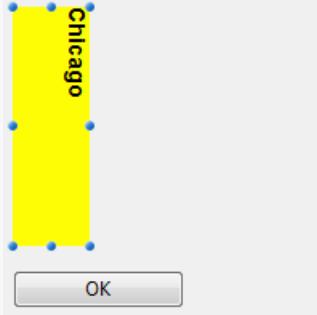
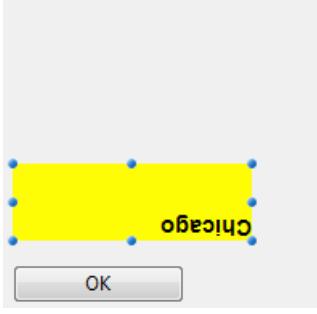
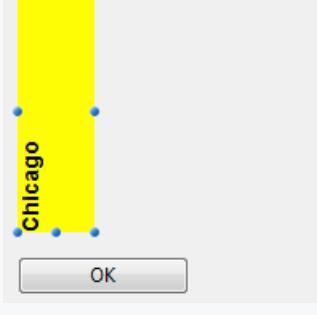
JSON 文法

名称	データタイプ	とりうる値
styledText	boolean	true, false

対象オブジェクト

方向

テキストエリアの角度 (回転) を変更します。テキストエリアは、90°単位で回転させることができます。それぞれの回転角度を適用するとき、オブジェクトの左下の角は固定されたままで回転していきます：

回転角度	戻り値
0 (デフォルト)	 <input type="button" value="OK"/>
90	 <input type="button" value="OK"/>
180	 <input type="button" value="OK"/>
270	 <input type="button" value="OK"/>

スタティックなテキストエリア のほかに、[入力不可](#) に設定された [入力オブジェクト](#) も回転させることができます。入力オブジェクトの方向プロパティにて 0°以外のオプションを選んだ場合、入力可プロパティは (選択されていた場合) 自動的に解除されます。その際、このオブジェクトは入力順から自動的に除外されます。

JSON 文法

名称	データタイプ	とりうる値
textAngle	number	0, 90, 180, 270

対象オブジェクト

入力 (入力不可) - テキストエリア

行フォントカラー配列

配列型リストボックス

リストボックスの各行/セルにカスタマイズしたフォントカラーを適用するために使用します。

倍長整数型の配列の名前を入力しなければなりません。配列のそれぞれの要素はリストボックスの行 (あるいは列のセル) に対応します。つまりこの配列は、各列に関連づけられている配列と同じサイズでなければいけません。ここでは [SET RGB COLORS](#) テーマの定数を使用することができます。もし上のレベルで定義されている背景色をそのままセルに継承したい場合には、対応する配列の要素に -255 を渡します。

JSON 文法

名称	データタイプ	とりうる値
rowStrokeSource	string	倍長整数型配列の名前

対象オブジェクト

[リストボックス - リストボックス列](#)

行スタイル配列

配列型リストボックス

リストボックスの各行/セルにカスタマイズされた文字スタイルを適用するために使用します。

倍長整数型の配列の名前を入力しなければなりません。配列のそれぞれの要素はリストボックスの行 (あるいは列のセル) に対応します。つまりこの配列は、各列に関連づけられている配列と同じサイズでなければいけません。配列には、[Font Styles](#) テーマの定数を使用することができます (メソッドを使用しての入力も可能)。定数同士を足し合わせてスタイルを組み合わせることもできます。もし上のレベルで定義されているスタイルをそのままセルに継承したい場合には、対応する配列の要素に -255 を渡します。

JSON 文法

名称	データタイプ	とりうる値
rowStyleSource	string	倍長整数型配列の名前

対象オブジェクト

[リストボックス - リストボックス列](#)

スタイルタグを全て保存

このプロパティは [マルチスタイル](#) 入力エリアの場合にのみ提供されます。このオプションがチェックされている場合には、たとえ変更がおこなわれていなくても、エリアはテキストとともにスタイルタグを格納します。この場合、タグはデフォルトスタイルが適用されます。このオプションがチェックされていないと、変更されたスタイルタグのみが格納されます。

たとえば、以下のようにスタイルが変更されたテキストがあります:

What a beautiful day

このプロパティが無効な場合、エリアは更新されたスタイルのみを格納します。つまり、格納される内容は以下のようになります:

```
What a <SPAN STYLE="font-size:13.5pt">beautiful</SPAN> day!
```

同プロパティが有効な場合には、エリアはすべてのフォーマット情報を格納します。先頭の汎用タグはデフォルトスタイルを定義し、変更されたスタイルはネストされたタグに書き込まれます。格納される内容は以下のようになります：

```
<SPAN STYLE="font-family:'Arial';font-size:9pt;text-align:left;font-weight:normal;font-style:normal;text
```

JSON 文法

名称	データタイプ	とりうる値
storeDefaultStyle	boolean	true, false (デフォルト)

対象オブジェクト

入力

テキスト、ピクチャー

背景パス名

オブジェクトの背景として使用するピクチャーのパスを指定します。複数の状態を持つアイコンを持つオブジェクトの場合、背景ピクチャーにも同じ状態の数が適用されます。

パス名は、[スタイルピクチャーのパス名プロパティ](#)と同じように指定します。

JSON 文法

名称	データタイプ	とりうる値
customBackgroundPicture	string	POSIX シンタックスの相対パス。style プロパティの "custom" オプションと併用する必要があります。

対象オブジェクト

[カスタムボタン](#) - [カスタムチェックボックス](#) - [カスタムラジオボタン](#)

ボタンスタイル

ボタンの外観を設定します。スタイルによっては、特定のオプションが利用できなくなることもあります。

JSON 文法

名称	データタイプ	とりうる値
style	テキスト	"regular", "flat", "toolbar", "bevel", "roundedBevel", "gradientBevel", "texturedBevel", "office", "help", "circular", "disclosure", "roundedDisclosure", "custom"

対象オブジェクト

[ボタン](#) - [ラジオボタン](#) - [チェックボックス](#)

横方向マージン

ボタン内側の横方向のマージンサイズ(ピクセル単位)を指定します。マージンにより、ボタンアイコンとタイトルの領域を制限します。

背景ピクチャーに境界が含まれるような場合に、このパラメーターを利用します：

マージン指定	例題
マージンなし	
13 ピクセルのマージン	

このプロパティは [縦方向マージン](#) プロパティとの組み合わせで機能します。

JSON 文法

名称	データタイプ	とりうる値
customBorderX	number	"カスタム" スタイルで利用可。最小値: 0

対象オブジェクト

[カスタムボタン](#) - [カスタムチェックボックス](#) - [カスタムラジオボタン](#)

アイコンの場所

フォームオブジェクトに対するアイコンの配置を指定します。

JSON 文法

名称	データタイプ	とりうる値
iconPlacement	string	"none", "left", "right"

対象オブジェクト

[リストボックスヘッダー](#)

アイコンオフセット

ボタンクリック時のオフセット値をピクセル単位で指定します。

このプロパティを使用すると、指定したピクセル数だけボタンタイトルが右下へシフトされます。この機能により、ボタンのクリック時に独自の3D 効果を適用することができます。

JSON 文法

名称	データタイプ	とりうる値
customOffset	number	最小値: 0

対象オブジェクト

[カスタムボタン](#) - [カスタムチェックボックス](#) - [カスタムラジオボタン](#)

状態の数

このプロパティは [ボタン](#)、[チェックボックス](#)、[ラジオボタン](#) のアイコンとして使用されるピクチャーに含まれる状態の数を指定します。一般的にボタンアイコンは4つの状態（アクティブ、クリック、ロールオーバー、無効）を含んでいます。

一つの状態につき、一つの画像を割り当てます。ソースピクチャーでは、状態を表すアイコンは縦に並んでいなければなりません：



状態は次の順番で割り当てられます:

1. ボタン未クリック / チェックボックス未選択 (変数値=0)
2. ボタンクリック / チェックボックス選択 (変数値=1)
3. ロールオーバー
4. disabled

JSON 文法

名称	データタイプ	とりうる値
iconFrames	number	最小値: 1

対象オブジェクト

ボタン (ヘルプボタンを除く) - チェックボックス - ラジオボタン

ピクチャーパス名

オブジェクトのアイコンに使用するピクチャーのパスを指定します。

パス名は、[スタティックピクチャーのパス名プロパティ](#)と同じように指定します。

アクティブオブジェクトのアイコンとして使う場合、[状態の数](#) 変数に対応するよう、デザインされている必要があります。

JSON 文法

名称	データタイプ	とりうる値
icon	picture	POSIX シンタックスの相対パス、またはファイルシステムパス

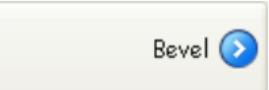
対象オブジェクト

ボタン (ヘルプボタンを除く) - チェックボックス - リストボックスヘッダー - ラジオボタン

タイトル/ピクチャー位置

このプロパティはアイコンに対するボタンタイトルの相対的な位置を指定します。ボタン内に、タイトルのみ (関連ピクチャーなし)、またはピクチャーのみ (タイトルなし) が含まれている場合、このプロパティは効果ありません。デフォルトでは、ピクチャーの下部にタイトルが置かれます。

このプロパティの各種オプションの結果を次に示します:

オプション	説明	例題
左	テキストはアイコンの左に置かれます。ボタンの内容は右揃えされます。	
上	テキストはアイコンの上に置かれます。ボタンの内容は中央揃えされます。	
右	テキストはアイコンの右に置かれます。ボタンの内容は左揃えされます。	
下	テキストはアイコンの下に置かれます。ボタンの内容は中央揃えされます。	
中央合わせ	アイコンのテキストはボタン内で縦と横に中央揃えされます。テキストをアイコンの中に組み込むような場合に利用します。	

JSON 文法

名称	データタイプ	とりうる値
textPlacement	string	"left", "top", "right", "bottom", "center"

対象オブジェクト

[ボタン \(ヘルプボタンを除く\)](#) - [チェックボックス](#) - [ラジオボタン](#)

縦方向マージン

ボタン内側の縦方向のマージンサイズ (ピクセル単位) を指定します。マージンにより、ボタンアイコンとタイトルの領域を制限します。

背景ピクチャーに境界が含まれるような場合に、このパラメーターを利用します。

このプロパティは [横方向マージン](#) プロパティとの組み合わせで機能します。

JSON 文法

名称	データタイプ	とりうる値
customBorderY	number	"カスタム" スタイルで利用可。最小値: 0

対象オブジェクト

[カスタムボタン](#) - [カスタムチェックボックス](#) - [カスタムラジオボタン](#)

ポップアップメニューあり

このプロパティを使用すると、ボタン内に逆三角形として表われるシンボルを表示することができます。このシンボルは、ポップアップメニューが付属することを示します:



このシンボルの外観と位置は、ボタンスタイルとプラットフォームによって変わります。

リンクと分離

ポップアップメニューシンボルをボタンに付加する際に、リンクと分離という2つのオプションから選択することができます:

リンク	分離

実際に "分離" モードを利用できるかどうかは、ボタンスタイルとプラットフォームによって決まります。

それぞれのオプションにより、ボタンとポップアップメニューとの関係が指定されます:

- ポップアップメニューが **分離** している場合、ボタンの左部分をクリックするとボタンのカレントアクションが直接実行されます。このアクションは、ボタンの右側からアクセスできるポップアップメニューを使用して変更することができます。
- ポップアップメニューが **リンク** している場合、ボタンをクリックしてもポップアップメニューが表示されるだけです。このポップアップメニュー上のアクションを選択しないと、実行はおこなわれません。

ポップアップメニューの管理

"ポップアップメニューあり" プロパティは、ボタンのグラフィック面だけを管理するという点に注意が必要です。ポップアップメニューとその値の表示は、すべて開発者が処理しなくてはなりません。具体的にはフォームイベントや [Dynamic pop up menu](#)、[Pop up menu](#) コマンドを使用してこれを処理します。

JSON 文法

名称	データタイプ	とりうる値
popupPlacement	string	<ul style="list-style-type: none">◦ "none"◦ "linked"◦ "separated"

対象オブジェクト

[ツールバー](#)ボタン - [ベベル](#)ボタン - [角の丸いベベル](#)ボタン - [OS X グラデーション](#)ボタン - [OS X テクスチャ](#)ボタン - [Office XP](#) ボタン - [サークル](#)ボタン - [カスタム](#)ボタン

Webエリア

4Dメソッドコールを許可

Webエリアで実行される JavaScript コードから 4D メソッドを呼び出して、戻り値を取得することができます。4D メソッドを Web エリアから呼び出せるようになるには、プロパティリストの "4D メソッドコールを許可" にチェックをする必要があります。

この機能は Web エリアが [埋め込み Web レンダリングエンジンを使用](#) している場合に限り、使用可能です。

このプロパティがチェックされている場合、特別な JavaScript オブジェクト `$4d` が Web エリア内にインスタンス化され、これを使用して [4D プロジェクトメソッドの呼び出しを管理](#) できるようになります。

JSON 文法

名称	データタイプ	とりうる値
methodsAccessibility	string	"none" (デフォルト), "all"

対象オブジェクト

Web エリア

進捗状況変数

倍長整数型変数の名前です。この変数には 0 から 100 までの値が格納され、この数値は Web エリアに表示されるページのロードされたパーセンテージを表します。この変数は 4D が自動で更新します。手動で変更することはできません。

As of 4D v19 R5, this variable is no longer updated in Web Areas using the [Windows system rendering engine](#).

JSON 文法

名称	データタイプ	とりうる値
progressSource	string	倍長整数型変数の名前

対象オブジェクト

Web エリア

URL

文字列型の変数で、Web エリアにロードされた URL またはロード中の URL が格納されます。変数と Web エリア間の連携は双方向でおこなわれます。

- ユーザーが新しい URL を変数に割り当てるとき、その URL は自動で Web エリアにロードされます。
- Web エリアでブラウズされると、自動で変数の内容が更新されます。

このエリアは Web ブラウザのアドレスバーのように機能します。Web エリアの上側にテキストエリアを置いて、内容を表示させることができます。

URL変数と WA OPEN URL コマンド

URL変数は [WA OPEN URL](#) コマンドと同じ効果をもたらします。しかしながら、以下の違いに注意してください。

- ドキュメントにアクセスする場合、この変数は RFC準拠 ("file:///c:/My%20Doc") な URL のみを受け付け、システムパス名 ("c:¥MyDoc") は受け付けません。[WA OPEN URL](#) コマンドは両方の記法を受け付けます。
- URL変数が空の文字列の場合、Webエリアは URL をロードしません。Webエリアがフォーム上で表示されていない場合（フォームの別ページに Webエリアがある場合等）、[WA OPEN URL](#) コマンドを実行しても効果はありません。
- [WA SET PAGE CONTENT](#): このコマンドを使用する場合、([WA OPEN URL](#) コマンドを呼び出すかあるいはエリアに割り当てられた URL変数への代入を通して) 少なくとも既に 1ページがエリア内に読み込まれている必要があります。
- URL変数がプロトコル (http, mailto, file など) を含まない場合、Webエリアは "http://" を付加します。[WA OPEN URL](#) コマンドはこれを付加しません。

JSON 文法

名称	データタイプ	とりうる値
urlSource	string	URL

対象オブジェクト

[Webエリア](#)

埋め込みWebレンダリングエンジンを使用

このオプションを使用して、Webエリアで使用する描画エンジンを 2つのうちから選択することができます:

- チェックなし - **JSON値: system** (デフォルト): この場合、4Dはシステムの最適なエンジンを使用します。この結果、HTML5 や JavaScript の最新 Web描画エンジンを自動的に利用できることになります。しかし、プラットフォーム間で若干描画に違いがでることがあります。Windows では、4Dは Microsoft Edge WebView2 を使用します。macOS では、カレントバージョンの WebKit (Safari) です。

Windows で Microsoft Edge WebView2がインストールされていない場合、4D はシステムのレンダリングエンジンとして埋め込みエンジンを使用します。システムにインストールされているかどうかを確認するには、アプリケーションパネルで "Microsoft Edge WebView2 Runtime" を検索してください。

- チェックあり - **JSON値: embedded** : この場合、4D は Chromium Embedded Framework (CEF) を使用します。埋め込みWebレンダリングエンジンを使用すると、Webエリアの描画とその動作が（ピクセル単位での若干の相違やネットワーク実装に関連する違いを除き）プラットフォームに関わらず同じになります。このオプションが選択されると、OS によりおこなわれる自動更新などの利点を得ることができなくなります。使用エンジンの新バージョンは 4D のリリースを通して定期的に提供されます。

CEFエンジンには以下のような制約があります:

- [WA SET PAGE CONTENT](#): このコマンドを使用する場合、([WA OPEN URL](#) コマンドを呼び出すかあるいはエリアに割り当てられた URL変数への代入を通して) 少なくとも既に 1ページがエリア内に読み込まれている必要があります。
- [WA SET PREFERENCE](#) コマンドの `WA enable URL drop` セレクターによって URLドロップが許可されている場合、最初のドロップをする前に少なくとも 1度は [WA OPEN URL](#) コマンドを呼び出すか、またはエリアに割り当てられている URL変数に URL が渡されている必要があります。

JSON 文法

名称	データタイプ	とりうる値
webEngine	string	"embedded", "system"

対象オブジェクト

[Webエリア](#)

On Activate

コード	呼び出し元	定義
11	フォーム	フォームウィンドウが最前面のウィンドウになった、またはサブフォームがフォーカスを得た

説明

フォームのウィンドウが背面に送られていた場合、そのウィンドウが最前面になったときにこのイベントが呼ばれます。

このイベントは個々のオブジェクトには適用されず、フォーム全体に適用されます。ゆえに `On Activate` フォームイベントプロパティが選択されれば、そのフォームのメソッドのみが呼び出されます。

サブフォームの場合には、コンテナーがフォーカスを得たとき ([フォーカス可](#) プロパティが設定されている場合) に、このイベントがサブフォームに渡されます。

On After Edit

コード	呼び出し元	定義
45	4D View Pro エリア - 4D Write Pro エリア - コンボボックス - フォーム - 入力 - 階層リスト - リストボックス - リストボックス列	フォーカスのある入力可能オブジェクトの内容が更新された

説明

一般的なケース

このイベントは、キーボード入力可能なオブジェクトへのデータ入力を最も低レベルでフィルターするために使用できます。

このイベントは、変更がおこなわれた方法に関係なく、入力可能オブジェクトの内容が変更されるたびに生成されます。つまり:

- ペーストやカット、削除、キャンセルなどの標準の編集アクション
- 値のドロップ (ペーストと同様のアクション)
- ユーザーがおこなったキーボードからの入力。この場合、`On After Edit` イベントは `On Before Keystroke` と `On After Keystroke` イベントの後に生成されます。
- ユーザーアクションをシミュレートするランゲージコマンドによる変更 (例: `POST KEY`)。

`On After Edit` イベント内において、入力テキストは `Get edited text` コマンドによって返されます。

4D View Pro

`FORM Event` によって返されるオブジェクトには以下のプロパティが格納されます:

プロパティ	タイプ	説明
code	倍長整数	<code>On After Edit</code>
description	テキスト	"On After Edit"
objectName	テキスト	4D View Pro エリア名
sheetName	テキスト	イベントが発生したシート名
action	テキスト	"editChange", "valueChanged", "DragDropBlock", "DragFillBlock", "formulaChanged", "clipboardPasted"

`action` プロパティの値に応じて、[イベントオブジェクト](#) には追加のプロパティが含まれます。

`action = editChange`

プロパティ	タイプ	説明
range	object	セルのレンジ
editingText	variant	カレントエディターでの値

`action = valueChanged`

プロパティ	タイプ	説明
range	object	セルのレンジ
oldValue	variant	変更前のセルの値
newValue	variant	変更後のセルの値

action = DragDropBlock

プロパティ	タイプ	説明
fromRange	object	ソースセルレンジ (ドラッグされる範囲) のレンジ
toRange	object	移行先セルレンジ (ドロップされる場所) のレンジ
copy	boolean	ソースレンジがコピーされたかどうかを表します
insert	boolean	ソースレンジが挿入されたかどうかを表します

action = DragFillBlock

プロパティ	タイプ	説明	fillDirection	longint	自動入力の方向
fillRange	object	自動入力のために使用されるレンジ			<ul style="list-style-type: none"> • 0: 全データ (値、書式、フォーミュラ) がセルに入力された • 1: 自動シーケンシャルデータがセルに入力された • 2: 書式のみがセルに入力された • 3: 値のみがセルに入力され、書式は入力されていない • 4: セルから値が除去された • 5: セルは自動的に入力された

action = formulaChanged

プロパティ	タイプ	説明
range	object	セルのレンジ
formula	テキスト	入力されたフォーミュラ

action = clipboardPasted

プロパティ	タイプ	説明
range	object	セルのレンジ
pasteOption	倍長整数	クリップボードから何をペーストされたかを表します: <ul style="list-style-type: none"> ● 0: すべて (値、書式、フォーミュラ) がペーストされた ● 1: 値のみがペーストされた ● 2: 書式のみがペーストされた ● 3: フォーミュラのみがペーストされた ● 4: 値と書式がペーストされた (フォーミュラはペーストされなかった) ● 5: フォーミュラと書式のみがペーストされた (値はペーストされなかった)
pasteData	object	クリップボードからペーストされるデータ <ul style="list-style-type: none"> ● "text" (テキスト): クリップボードからのテキスト ● "html" (テキスト): クリップボードからの HTML

例題

以下は `On After Edit` イベントを管理する例です:

```
If(FORM Event.code=On After Edit)
    If(FORM Event.action="valueChanged")
        ALERT("WARNING: You are currently changing the value\
            from "+String(FORM Event.oldValue)+\
            " to "+String(FORM Event.newValue)+"!")
    End if
End if
```

上記のコードにより生成されたイベントオブジェクトは、以下のような形式をしています:

```
{
  "code":45,
  "description":"On After Edit",
  "objectName":"ViewProArea",
  "sheetname":"Sheet1",
  "action":"valueChanged",
  "range": {area:ViewProArea,ranges:[{column:1,row:2,sheet:1}]},
  "oldValue":"The quick brown fox";
  "newValue":"jumped over the lazy dog";
}
```

On After Keystroke

コード	呼び出し元	定義
28	4D Write Pro エリア - コンボボックス - フォーム - 入力 - リストボックス - リストボックス列	フォーカスのあるオブジェクトに文字が入力されようとしている。 <code>Get edited text</code> はこの文字を含む オブジェクトのテキストを返します。

▶ 履歴

説明

`On After Keystroke` イベントは、一般的に `On After Edit` イベントで置き換えることができます（後述参照）。

`On Before Keystroke` と `On After Keystroke` イベントプロパティを選択すると、`FORM Event` コマンドを使用して返される `On Before Keystroke` と `On After Keystroke` イベントを検知し、オブジェクトへのキーストロークを処理できます（詳細は `Get edited text` コマンドの説明を参照ください）。

これらのイベントは `POST KEY` のようなユーザーアクションをシミュレートするコマンドによって生成されます。

`On After Keystroke` イベントは次の場合には生成されません：

- `リストボックス列` メソッドの場合、ただし、セルを編集している場合を除きます（`リストボックス` メソッドではどのような場合でも生成されます）。
- キーボードを使用せずに（ペーストやドラッグ & ドロップ、チェックボックス、ドロップダウンリスト、コンボボックス）おこなわれた変更の場合。これらのイベントを処理するには `On After Edit` を使用します。

キーストロークシーケンス

入力に一連のキーストロークが必要な場合、`On Before Keystroke` と `On After Keystroke` イベントは、入力がユーザーによって完全に確定されたときにのみ生成されます。`Keystroke` コマンドは、確定済みの文字を返します。このケースは主に以下のように発生します：

- ^ や ~ のような特殊キーが使用された場合：その後に拡張された文字が入力された場合にのみ生成されます（例： "ê" や "ñ"）。
- IME (Input Method Editor) が、文字の組み合わせを入力するための中間的なダイアログボックスを表示している場合：IME のダイアログが確定されたときにのみイベントが発生します。

参照

[On Before Keystroke](#)。

On After Sort

コード	呼び出し元	定義
30	リストボックス - リストボックス列	リストボックス列内で標準のソートがおこなわれた

説明

このイベントは標準の並べ替えがおこなわれた直後に生成されます（ただし [On Header Click](#) イベントで \$0 に -1 が返された場合には生成されません）。このメカニズムは、ユーザーによっておこなわれた直近の並べ替えの方向を保存するのに使用できます。このイベント内で `Self` コマンドは、並べ替えられたカラムヘッダー変数へのポインターを返します。

On Alternative Click

コード	呼び出し元	定義
38	ボタン - リストボックス - リストボックス列	<ul style="list-style-type: none">ボタン: ボタンの "矢印" のエリアがクリックされたリストボックス: オブジェクト配列のカラムにおいて、エリプシスボタン ("alternateButton" 属性) がクリックされた

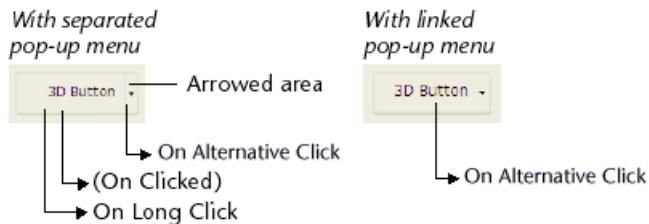
説明

ボタン

いくつかのボタンスタイルには、[ポップアップメニュー](#) をリンクし、矢印を表示させることができます。この矢印をクリックすると、ボタンの主たるアクションの代わりのアクションを提供するポップアップを表示します。

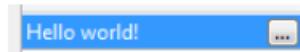
4D では `On Alternative Click` イベントを使用してこの動作を管理できます。このイベントは、ユーザーが矢印をクリックすると、マウスボタンが押されてすぐに生成されます：

- ポップアップメニューが 分離 されている場合、このイベントはボタン中で矢印のあるエリアがクリックされた場合のみ生成されます。
- ポップアップメニューが リンク されている場合、このイベントはボタン上どこをクリックしても生成されます。このタイプのボタンでは `On Long Click` イベントが生成されないことに注意してください。



リストボックス

このイベントは [オブジェクト配列型のリストボックス](#) のカラムにおいて、ユーザーがウィジェットのエリプシスボタン ("alternateButton" 属性) をクリックしたときに生成されます。



["alternateButton" 属性の説明](#) を参照ください。

On Before Data Entry

コード	呼び出し元	定義
41	リストボックス - リストボックス列	リストボックスセルが編集モードに変更されようとしている

説明

このイベントは、リストボックス中のセルが編集される直前に生成されます（入力カーソルが表示される前）。このイベントを使用して、たとえば表示中と編集中で異なるテキストを表示させることができます。

カーソルがセルに入ると、そのリストボックスまたは列のメソッドで `On Before Data Entry` イベントが生成されます。

- このイベントのコンテキストにおいて、\$0 に -1 を設定すると、そのセルは入力不可として扱われます。Tab や Shift+Tab が押された後にイベントが生成された場合には、フォーカスはそれぞれ次あるいは前のセルに移動します。
- \$0 が -1 でなければ（デフォルトは 0）、列は入力可であり編集モードに移行します。

[入力の管理](#) の章を参照ください。

On Before Keystroke

コード	呼び出し元	定義
17	4D Write Pro エリア - コンボボックス - フォーム - 入力 - リストボックス - リストボックス列	フォーカスのあるオブジェクトに文字が入力されようとしている。 <code>Get edited text</code> はこの文字を含まないオブジェクトのテキストを返します。

▶ 履歴

説明

`On Before Keystroke` と `On After Keystroke` イベントプロパティを選択すると、`FORM Event` コマンドを使用して返される `On Before Keystroke` と `On After Keystroke` イベントを検知し、オブジェクトへのキーストロークを処理できます（詳細は `Get edited text` コマンドの説明を参照ください）。`On Before Keystroke` イベント内では、`FILTER KEYSTROKE` コマンドを使って、入力された文字をフィルターできます。

これらのイベントは `POST KEY` のようなユーザーアクションをシミュレートするコマンドによっても生成されます。

`On Before Keystroke` イベントは次の場合には生成されません：

- `リストボックス列` メソッドの場合、ただし、セルを編集している場合を除きます（`リストボックス` メソッドではどのような場合でも生成されます）。
- キーボードを使用せずに（ペーストやドラッグ & ドロップ、チェックボックス、ドロップダウンリスト、コンボボックス）おこなわれた変更の場合。これらのイベントを処理するには `On After Edit` を使用します。

入力不可オブジェクト

`On Before Keystroke` イベントは、入力不可能なオブジェクトでも発生させることができます。たとえば、リストボックスのセルが入力不可能であったり、行が選択不可能であったりしても、リストボックスで発生させることができます。これにより、ユーザーが値の最初の文字を入力することで、リストボックスの特定の行に動的にスクロールできるようなインターフェースを構築することができます。リストボックスのセルが入力可能な場合は、`Is editing text` コマンドを使用して、ユーザーが実際にセルにテキストを入力しているのか、タイプアヘッド機能を使用しているのかを確認し、適切なコードを実行することができます。

キーストロークシーケンス

入力に一連のキーストロークが必要な場合、`On Before Keystroke` と `On After Keystroke` イベントは、入力がユーザーによって完全に確定されたときにのみ生成されます。`Keystroke` コマンドは、確定済みの文字を返します。このケースは主に以下のように発生します：

- ^ や ~ のような特殊キーが使用された場合：その後に拡張された文字が入力された場合にのみ生成されます（例："ê" や "ñ"）。
- IME (Input Method Editor) が、文字の組み合わせを入力するための中間的なダイアログボックスを表示している場合：IME のダイアログが確定されたときにのみイベントが発生します。

参照

[On After Keystroke](#)。

On Begin Drag Over

コード	呼び出し元	定義
17	4D Write Pro エリア - ボタン - ボタングリッド - チェックボックス - ドロップダウンリスト - フォーム - 階層リスト - 入力 - リストボックス - リストボックス列 - ピクチャーボタン - ピクチャーポップアップメニュー - プラグインエリア - 進捗インジケーター - ラジオボタン - ルーラー - スピナー - スプリッター - ステッパー - タブコントロール	オブジェクトがドラッグされている

説明

`On Begin Drag Over` フォームイベントは、ドラッグ可能なすべてのフォームオブジェクトで選択できます。このイベントは、オブジェクトが **ドラッグ有効** プロパティを持っているすべてのケースで生成されます。このイベントは、ソースオブジェクトのメソッドまたはソースオブジェクトのフォームメソッドから呼び出すことができます。

`On Drag Over` フォームイベントとは異なり、`On Begin Drag Over` イベントはドラッグアクションの ソースオブジェクト のコンテキスト内で呼び出されます。

`On Begin Drag Over` イベントは、ドラッグアクションの準備に役立ちます。このイベントは以下のように使用できます:

- `APPEND DATA TO PASTEBOARD` コマンドを使って、ペーストボードにデータや署名を追加する。
- `SET DRAG ICON` コマンドを使って、ドラッグアクション中にカスタムアイコンを表示する。
- ドラッグされたオブジェクトのメソッドの `$0` を使用して、ドラッグを許可/拒否する。
 - ドラッグアクションを受け入れるには、ソースオブジェクトのメソッドは (`$0:=0` を実行して) `0` (ゼロ) を返さなければなりません。
 - ドラッグアクションを拒否するには、ソースオブジェクトのメソッドは (`$0:=-1` を実行して) `-1` (マイナス1) を返さなければなりません。
 - 結果が返されない場合は、ドラッグアクションが受け入れられたと 4D は判断します。

4D のデータは、イベントが呼び出される前に、ペーストボードに置かれます。たとえば、自動ドラッグ アクションなしでドラッグした場合、ドラッグされたテキストは、イベントが呼び出される時にはペーストボードにあります。

On Begin URL Loading

コード	呼び出し元	定義
47	Webエリア	新しい URL が Web エリアにロードされた

説明

このイベントは、Webエリアに新しい URL のロードを開始した時に生成されます。Webエリアに関連付けられた `URL` 変数を使用してロード中の URL をることができます。

ロード中の URL は [カレントURL](#) とは異なります (`WA Get current URL` コマンドの説明参照)。

On Bound Variable Change

コード	呼び出し元	定義
54	フォーム	サブフォームにバインドされた変数が更新された

説明

このイベントは、親フォーム中のサブフォームにバインドされた変数の値が更新され（同じ値が代入された場合でも）、かつサブフォームがカレントフォームページまたはページ0に属している場合に、[サブフォーム](#) のフォームメソッドのコンテキストで生成されます。

詳細について、[バインドされた変数の管理](#) を参照してください。

On Clicked

コード	呼び出し元	定義
4	4D View Pro エリア - 4D Write Pro エリア - ボタン - ボタングリッド - チェックボックス - コンボボックス - ドロップダウンリスト - フォーム - 階層リスト - 入力 - リストボックス - リストボックス列 - ピクチャーボタン - ピクチャーポップアップメニュー - プラグインエリア - 進捗インジケーター - ラジオボタン - ルーラー - スピナー - スプリッター - ステッパー - タブコントロール	オブジェクト上でクリックされた

説明

`On Clicked` イベントは、ユーザーがオブジェクト上でクリックしたときに発生します。

いくつかのフォームオブジェクトはキーボードからも操作可能です。たとえば、チェックボックスがフォーカスを得ると、スペースバーでオン/オフを切り替えることができます。この場合でも `On Clicked` イベントは生成されます。

`On Clicked` イベントは通常、マウスボタンが離されたときに生成されます。しかし、いくつか例外があります:

- **非表示ボタン:** マウスがクリックされると、ボタンが離されるのを待たずに `On Clicked` イベントが生成されます。
- **ルーラー:** `オブジェクトメソッド実行` オプションが `true` に設定されていると、`On Clicked` イベントはクリックがおこなわれるとすぐに生成されます。
- **コンボボックス:** `On Clicked` イベントは、割り当てられたメニューでユーザーが別の値を選択した場合にのみ発生します。`コンボボックス` は、割り当てられたドロップダウンリストにデフォルト値が提供された、入力可能なテキストエリアとして扱われます。つまり、コンボボックス内におけるデータ入力処理は、`On Before Keystroke` や `On After Keystroke`、`On Data Change` イベントを使用しておこなう必要があります。
- **ドロップダウンリスト:** `On Clicked` イベントは、ユーザーがメニューで別の値を選択した場合にのみ発生します。`On Data Change` イベントは、現在の値とは異なる値が選択されたときに、オブジェクトが操作されたことを検出することができます。
- リストボックスの入力セルが `編集中` のとき、マウスボタンが押されると `On Clicked` イベントが発生するので、`Contextual click` コマンドなどを使用することができます。

`On Clicked` イベントのコンテキストにおいては `Clickcount` コマンドを使うことによってユーザーがおこなったクリック数をテストすることができます。

On Clicked と On Double Clicked

`On Clicked` や `On Double Clicked` オブジェクトイベントプロパティを選択したのち、`FORM Event` コマンドを使用してオブジェクト上でのクリックを検知し処理することができます。`FORM Event` コマンドはユーザーアクションに応じ、`On Clicked` または `On Double Clicked` を返します。

両イベントがオブジェクトに対し選択されている場合、ダブルクリックがおこなわれるとまず `On Clicked` が、そして `On Double Clicked` イベントが生成されます。

4D View Pro

このイベントは、4D View Pro ドキュメント上でクリックが発生したときに生成されます。このコンテキストにおいて、`FORM Event` コマンドによって返される `イベントオブジェクト` には以下のプロパティが含まれています:

プロパティ	タイプ	説明
code	倍長整数	<code>On Clicked</code>
description	テキスト	"On Clicked"
objectName	テキスト	4D View Pro エリア名
sheetName	テキスト	イベントが発生したシート名
range	object	セルのレンジ

例題

```
If(FORM Event.code=On Clicked)
    VP SET CELL STYLE(FORM Event.range;New object("backColor";"green"))
End if
```

On Close Box

コード	呼び出し元	定義
22	フォーム	ウィンドウのクローズボックスがクリックされた

説明

On Close Box イベントは、ユーザーがウィンドウのクローズボックスをクリックすると生成されます。

例題

この例題では、レコードのデータ入力に使われるフォームで、ウィンドウを閉じるイベントを処理する方法を示します：

```
// 入力フォームのメソッド
$vpFormTable:=Current form table
Case of
//...
:(Form event code=On Close Box)
If(Modified record($vpFormTable->))
    CONFIRM("レコードが変更されました。 へんこうを保存しますか？")
    If(OK=1)
        ACCEPT
    Else
        CANCEL
    End if
Else
    CANCEL
End if
//...
End case
```

On Close Detail

コード	呼び出し元	定義
26	フォーム - リストボックス	入力フォームから離れ、出力フォームに戻ろうとしている

説明

`On Close Detail` イベントは次のコンテキストで利用できます:

- 出力フォーム: このイベントは、詳細フォームが閉じられ、ユーザーがリストフォームに戻るときに生成されます。このイベントは、プロジェクトフォームでは選択できず、テーブルフォームでのみ利用できます。
- セレクション型 のリストボックス: このイベントはセレクション型リストボックスに関連付けられた 詳細フォーム に表示されたレコードが閉じられるときに生成されます（レコードが変更されたかどうかは関係しません）。

On Collapse

コード	呼び出し元	定義
44	階層リスト - リストボックス	クリックやキーストロークで階層リストまたは階層リストボックスの要素が折りたたまれた

説明

- [階層リスト](#): このイベントは、マウスクリックやキーストロークで階層リストの要素が折りたたまれるたびに呼び出されます。
- [階層リストボックス](#): このイベントは、階層リストボックスの行が折りたたまれたときに生成されます。

参照

[On Expand](#)

On Column Moved

コード	呼び出し元	定義
32	リストボックス - リストボックス列	リストボックスの列がユーザーのドラッグ & ドロップで移動された

説明

このイベントは、ユーザーのドラッグ & ドロップでリストボックスの列が移動されたときに生成されます（[許可されている場合](#)）。ただし、元の場所にドロップされた場合には生成されません。

`LISTBOX MOVED COLUMN NUMBER` コマンドは列の新しい位置を返します。

On Column Resize

コード	呼び出し元	定義
33	4D View Pro エリア - リストボックス - リストボックス列	ユーザーのマウス操作によって、またはフォームウィンドウのリサイズによって、カラムの幅が変更された

説明

リストボックス

このイベントは、ユーザーによってリストボックスの列幅が変更されたときに生成されます。このイベントは "ライブ" にトリガーされます。つまり、対象となるリストボックスあるいはカラムがリサイズされている間はずっと継続して送信されづけます。リサイズはユーザーによって手動でおこなわれるか、あるいはフォームウィンドウ自身のリサイズの結果リストボックスとそのカラムがリサイズされる場合も含みます（手動によるフォームのリサイズおよび RESIZE FORM WINDOW コマンドを使用したリサイズ）。

余白カラム がリサイズされた場合には、 On Column Resize イベントはトリガーされません

4D View Pro

このイベントはカラムの幅がユーザーによって変更されたときに生成されます。このコンテキストにおいて、 FORM Event コマンドによって返される イベントオブジェクト には以下のプロパティが含まれています：

プロパティ	タイプ	説明
code	倍長整数	On Column Resize
description	テキスト	"On Column Resize"
objectName	テキスト	4D View Pro エリア名
sheetName	テキスト	イベントが発生したシート名
range	object	幅が変更されたカラムのセルレンジ
headers	boolean	行ヘッダーカラム（最初のカラム）がリサイズされた場合には true、それ以外の場合には false

例題

```
If(FORM Event.code=On Column Resize)
  VP SET CELL STYLE(FORM Event.range;New object("hAlign";vk horizontal align right))
End if
```

On Data Change

コード	呼び出し元	定義
20	4D Write Pro エリア - ドロップダウンリスト - フォーム - 階層リスト - 入力 - リストボックス - リストボックス列 - プラグインエリア - 進捗インジケーター - ルーラー - スピナー - ステッパー - サブフォーム	オブジェクトのデータが変更された

説明

`On Data Change` イベントプロパティがオブジェクトで選択されている場合、`FORM Event` コマンドを使って、データソースの値の変化を検出し、処理することができます。

イベントは、オブジェクトに結び付けられた変数が 4D により内部的に更新され次第、生成されます（一般的には、入力エリアオブジェクトがフォーカスを失った時）。

サブフォームにおいては、`On Data Change` イベントはサブフォームオブジェクト変数の値が更新されたときにトリガーされます。

On Deactivate

コード	呼び出し元	定義
12	フォーム	フォームウィンドウが最前面のウィンドウでなくなった

説明

フォームのウィンドウが最前面にあった場合、そのウィンドウが背面に送られたときにこのイベントが呼ばれます。

このイベントは個々のオブジェクトには適用されず、フォーム全体に適用されます。ゆえに `On Deactivate` フォームイベントプロパティが選択されれば、そのフォームのメソッドのみが呼び出されます。

参照

[On Activate](#)

On Delete Action

コード	呼び出し元	定義
58	階層リスト - リストボックス	ユーザーが項目の削除を試みた

説明

このイベントは、ユーザーが削除キー（Delete や Backspace キー）を押して、またはクリア標準アクションが割り当てられたメニュー項目（編集メニューの クリア 等）を選択して、選択された項目の削除を指示したときに生成されます。

4D はイベントの生成だけをおこなうことに留意してください。4D は項目を消去しません。実際の削除処理や事前警告の表示などは開発者の責任です。

On Display Detail

コード	呼び出し元	定義
8	フォーム - リストボックス	レコードがリストフォーム中に、あるいは行がリストボックス中に表示されようとしている

説明

`On Display Detail` イベントは次のコンテキストで利用できます:

出力フォーム

`DISPLAY SELECTION` や `MODIFY SELECTION` によって、リストフォームでレコードを表示されようとしています。

このイベントは、プロジェクトフォームでは選択できず、テーブルフォームでのみ利用できます。

このコンテキストにおいて、メソッドやフォームイベントが呼び出される順序は以下のとおりです:

- レコードごとに:
 - 詳細エリアのオブジェクトごとに:
 - オブジェクトメソッドの `On Display Detail` イベント
 - フォームメソッドの `On Display Detail` イベント

ヘッダー領域は、`On Header` イベントで処理されます。

`On Display Detail` イベントから、ダイアログボックスを表示する 4Dコマンドを呼び出すことはできません。これはシンタックスエラーを起こします。以下のコマンドが該当します: `ALERT`, `DIALOG`, `CONFIRM`, `Request`, `ADD RECORD`, `MODIFY RECORD`, `DISPLAY SELECTION`, `MODIFY SELECTION`。

セレクションリストボックス

このイベントは、[セレクション型](#) のリストボックスの行が表示されたときに発生します。

Displayed line number

`Displayed line number` 4Dコマンドは、`On Display Detail` フォームイベントと連動します。このコマンドは、レコードのリストまたはリストボックスの行が画面に表示されるときに処理されている行の番号を返します。

On Double Clicked

コード	呼び出し元	定義
13	4D View Pro エリア - 4D Write Pro エリア - ボタン - ボタングリッド - チェックボックス - コンボボックス - ドロップダウンリスト - フォーム - 階層リスト - 入力 - リストボックス - リストボックス列 - ピクチャーボタン - ピクチャーポップアップメニュー - プログラムエディタ - 進捗インジケーター - ラジオボタン - ルーラー - スピナー - スプリッター - ステッパー - タブコントロール	オブジェクト上でダブルクリックされた

説明

`On Double Clicked` イベントは、ユーザーがオブジェクトをダブルクリックしたときに発生します。ダブルクリック間隔の最大時間は、システム環境設定で定義されています。

`On Clicked` や `On Double Clicked` オブジェクトイベントプロパティを選択したのち、`FORM Event` コマンドを使用してオブジェクト上でのクリックを検知し処理することができます。`FORM Event` コマンドはユーザーアクションに応じ、`On Clicked` または `On Double Clicked` を返します。

両イベントがオブジェクトに対し選択されている場合、ダブルクリックがおこなわれるとまず `On Clicked` が、そして `On Double Clicked` イベントが生成されます。

4D View Pro

このイベントは、4D View Pro ドキュメント上でダブルクリックが発生したときに生成されます。このコンテキストにおいて、`FORM Event` コマンドによって返される `イベントオブジェクト` には以下のプロパティが含まれています：

プロパティ	タイプ	説明
code	倍長整数	13
description	テキスト	"On Double Clicked"
objectName	テキスト	4D View Pro エリア名
sheetName	テキスト	イベントが発生したシート名
range	object	セルのレンジ

例題

```
If(FORM Event.code=On Double Clicked)
  $value:=VP Get value(FORM Event.range)
End if
```

On Drag Over

コード	呼び出し元	定義
21	4D Write Pro エリア - ボタン - ボタングリッド - チェックボックス - ドロップダウンリスト - 階層リスト - 入力 - リストボックス - リストボックス列 - ピクチャーボタン - ピクチャーポップアップメニュー - ブラグインエリア - 進捗インジケーター - ラジオボタン - ルーラー - スピナー - スプリッター - ステッパー - タブコントロール	データがオブジェクト上にドロップされる可能性がある

説明

`On Drag Over` イベントは、マウスポインターがオブジェクトの上を移動する時に、繰り返しドロップ先オブジェクトに送られます。このイベントの応答として、開発者は通常、以下のことをおこないます：

- ペーストボード内にあるデータや署名を (`GET PASTEBOARD DATA` コマンドを使用して) 取得する。
- ペーストボードのデータの状態や型に基づき、ドラッグ & ドロップの受け付けまたは拒否をおこないます。

ドラッグを受け付けるには、ドロップ先のオブジェクトメソッドが (`$0:=0` を実行して) 0 (ゼロ) を返さなければなりません。ドラッグを拒否するには、オブジェクトメソッドが (`$0:=-1` を実行して) -1 (マイナス1) を返さなければなりません。`On Drag Over` イベント中、4D はこのオブジェクトメソッドを関数として扱います。結果が返されない場合には、4D はドラッグが受け付けられたものと認識します。

ドラッグを受け入れると、ドロップ先オブジェクトがハイライトされます。ドラッグを拒否した場合、ドロップ先オブジェクトはハイライトされません。ドラッグを受け付けることは、ドラッグされたデータがドロップ先オブジェクトに挿入されるという意味ではありません。これは、単にマウスボタンをこの場所で離したときに、ドラッグされたデータがドロップ先オブジェクトによって受け付けられ、`On Drop` イベントが動くということを意味するだけです。

ドロップ可能なオブジェクトに対して開発者が `On Drag Over` イベントを処理しない場合には、そのオブジェクトは、ドラッグされたデータの性質やタイプに関係なく、すべてのドラッグ処理に対してハイライトされます。

`On Drag Over` イベントは、ドラッグ & ドロップ処理の最初の段階を制御する手段です。ドラッグされたデータがドロップ先オブジェクトと互換性のあるタイプかどうかをテストでき、またドラッグの受け付けや拒否をできるだけでなく、4D があなたの判断に基づいてドロップ先オブジェクトをハイライト (または無反応) されるため、この操作が有効であることを操作者にフィードバックすることができます。

`On Drag Over` イベントはマウスの移動に従って、現在のドロップ先オブジェクトに対して繰り返し送られるため、このイベントのコード処理は短く、短時間で実行されるようにしてください。

参照

[On Begin Drag Over](#)

On Drop

コード	呼び出し元	定義
16	4D Write Pro エリア - ボタン - ボタングリッド - チェックボックス - ドロップダウンリスト - フォーム - 階層リスト - 入力 - リストボックス - リストボックス列 - ピクチャーボタン - ピクチャーポップアップメニュー - プラグインエリア - 進捗インジケーター - ラジオボタン - ルーラー - スピナー - スプリッター - ステッパー - タブコントロール	データがオブジェクトにドロップされた

説明

`On Drop` イベントはマウスポインターがドロップ先オブジェクト上でリリースされたときにそのオブジェクトに一度送られます。このイベントはドラッグ & ドロップ処理の第2段階であり、ユーザーアクションの応答として処理を実行します。

このイベントは、`On Drag Over` イベント中にドラッグが受け付けられなかった場合には、オブジェクトに送られません。オブジェクトに対して `On Drag Over` イベントを処理し、ドラッグを拒否した場合には、`On Drop` イベントは発生しません。つまり、`On Drag Over` イベント中にソースオブジェクトとドロップ先オブジェクト間のデータタイプの互換性をテストして、有効なドロップを受け付けた場合には、`On Drop` 中にデータの再テストをする必要はありません。データがドロップ先オブジェクトに対して適切であることは既にわかっているためです。

参照

[On Begin Drag Over](#)

On End URL Loading

コード	呼び出し元	定義
49	Webエリア	URL のすべてのリソースがロードされた

説明

このイベントは、現在の URL のすべてのリソースがロードが完了すると生成されます。 `WA Get current URL` コマンドを使用して、ロードされた URL をることができます。

On Expand

コード	呼び出し元	定義
44	階層リスト - リストボックス	クリックやキーストロークで階層リストまたは階層リストボックスの要素が展開された

説明

- [階層リスト](#): このイベントは、マウスクリックやキーストロークで階層リストの要素が展開されるたびに呼び出されます。
- [階層リストボックス](#): このイベントは、階層リストボックスの行が展開されたときに生成されます。

参照

[On Collapse](#)

On Footer Click

コード	呼び出し元	定義
57	リストボックス - リストボックス列	リストボックス列のフッターがクリックされた

説明

このイベントはリストボックスやリストボックス列で利用できます。このイベントは、リストボックスやリストボックス列のフッターエリアがクリックされたときに生成されます。この場合、`OBJECT Get pointer` コマンドはクリックされたフッター変数へのポインターを返します。イベントは左および右クリックどちらでも生成されます。

`Clickcount` コマンドを使うことによってユーザーがおこなったクリック数をテストすることができます。

On Getting focus

コード	呼び出し元	定義
15	4D View Pro エリア - 4D Write Pro エリア - ボタン - チェックボックス - コンボボックス - フォーム - 階層リスト - 入力 - リストボックス - リストボックス列 - プラグインエリア - 進捗インジケーター - ラジオボタン - ルーラー - スピナー - ステッパー - サブフォーム - Webエリア	フォームオブジェクトがフォーカスを得た

説明

`On Getting Focus` イベントや `On Losing Focus` イベントを使って、[フォーカス可](#) オブジェクトのフォーカスの変更を処理できます。

[サブフォームオブジェクト](#) の場合、このイベントがプロパティリスト中でチェックされていれば、サブフォームオブジェクトのメソッド内で生成されます。このイベントは、サブフォームのフォームメソッドに送信されます。これによってたとえば、フォーカスに応じてサブフォーム中のナビゲーションボタンの表示を管理できます。サブフォームオブジェクトは、それ自体がフォーカスを持つ点に留意してください。

On Header

コード	呼び出し元	定義
5	4D Write Pro エリア - ボタン - ボタングリッド - チェックボックス - ドロップダウンリスト - フォーム (リストフォームのみ) - 階層リスト - 入力 - ピクチャーボタン - ピクチャーポップアップメニュー - プラグインエリア - 進捗インジケーター - ラジオボタン - ルーラー - スピナー - スプリッター - ステッパー - タブコントロール	フォームのヘッダーエリアが印刷あるいは表示されようとしている

説明

`On Header` イベントは、`DISPLAY SELECTION` や `MODIFY SELECTION` によって、リストフォームでレコードを表示されようとしているときに呼び出されます。

このイベントは、プロジェクトフォームでは選択できず、テーブルフォーム でのみ利用できます。

このコンテキストにおいて、メソッドやフォームイベントが呼び出される順序は以下のとおりです:

- ヘッダーエリアのオブジェクトごとに:
 - オブジェクトメソッドの `On Header` イベント
 - フォームメソッドの `On Header` イベント

印刷されるレコードは、`On Display Detail` イベントで処理されます。

`On Header` イベントから、ダイアログボックスを表示する 4Dコマンドを呼び出すことはできません。これはシンタックスエラーを起こします。以下のコマンドが該当します: `ALERT` , `DIALOG` , `CONFIRM` , `Request` , `ADD RECORD` , `MODIFY RECORD` , `DISPLAY SELECTION` , `MODIFY SELECTION` 。

On Header Click

コード	呼び出し元	定義
42	4D View Pro エリア - リストボックス - リストボックス列	リストボックスの列ヘッダーでクリックがおこなわれた

説明

リストボックス

このイベントは、リストボックスの列ヘッダーでクリックがおこなわれると生成されます。この場合 `Self` コマンドを使用すればクリックされた列ヘッダーを知ることができます。

リストボックスで `ソート可` プロパティが選択されている場合、`$0` に 0 または -1 を渡して標準の並べ替えをおこなうかどうか指定できます：

- `$0 = 0` の場合、標準の並べ替えがおこなわれます。
- `$0 = -1` の場合、標準の並べ替えはおこなわれず、ヘッダーには並べ替え矢印は表示されません。開発者は 4D ランゲージを使用して、カスタマイズされた条件に基づく並べ替えを実行できます。

リストボックスで `ソート可` プロパティが選択されていない場合、`$0` は使用されません。

4D View Pro

このイベントは、4D View Pro ドキュメント内のカラムヘッダーまたは行ヘッダーでクリックが発生したときに生成されます。このコンテキストにおいて、`FORM Event` コマンドによって返される `イベントオブジェクト` には以下のプロパティが含まれています：

プロパティ	タイプ	説明
code	倍長整数	42
description	テキスト	"On Header Click"
objectName	テキスト	4D View Pro エリア名
sheetName	テキスト	イベントが発生したシート名
range	object	セルのレンジ
sheetArea	倍長整数	イベントが発生したシートの場所： <ul style="list-style-type: none">• 0：カラム文字ヘッダー / 行番号のヘッダーの間の交差領域（シートの左上）• 1：カラムヘッダー（カラム文字を示す領域）• 2：行ヘッダー（行番号を示す領域）

例題

```
If(FORM Event.code=On Header Click)
  Case of
    :(FORM Event.sheetArea=1)
      $values:=VP Get values(FORM Event.range)
    :(FORM Event.sheetArea=2)
      VP SET CELL STYLE(FORM Event.range;New object("backColor";"gray"))
    :(FORM Event.sheetArea=0)
      VP SET CELL STYLE(FORM Event.range;New object("borderBottom";\
        New object("color";"#800080";"style";vk line style thick)))
  End case
End if
```


On Load

コード	呼び出し元	定義
1	4D View Pro エリア - 4D Write Pro エリア - ボタン - ボタングリッド - チェックボックス - コンボボックス - ドロップダウントラスト - フォーム - 階層リスト - 入力 - リストボックス - リストボックス列 - ピクチャーボタン - ピクチャーポップアップメニュー - プラグインエリア - 進捗インジケーター - ラジオボタン - ルーラー - スピナー - スプリッター - ステッパー - サブフォーム - タブコントロール - Webエリア	フォームが表示または印刷されようとしている

説明

このイベントは、フォームが表示または印刷されようとしているときに生成されます。

`On Load` オブジェクトイベントプロパティが選択されている、フォームの全ページの全オブジェクトのオブジェクトメソッドが呼び出されます。その後、`On Load` フォームイベントプロパティが選択されていれば、フォームメソッドが呼び出されます。

オブジェクトの `On Load` と `On Unload` イベントが生成されるには、オブジェクトとオブジェクトが属するフォームの両方で有効にされていなければなりません。オブジェクトのみでイベントが有効になっている場合、イベントは生成されません。これら 2つのイベントはフォームレベルでも有効にされていなければなりません。

サブフォーム

`On Load` イベントは、サブフォームを開く際に生成されます。これらのイベントは親フォームレベルでも有効にされていなければなりません。このイベントは、親フォームのイベントよりも前に生成される点に留意してください。また、フォームイベント動作の原則に従い、サブフォームが 0 もしくは 1 以外のページに配置されている場合、このイベントはページが開かれるときに生成され、フォームが開かれるときではないことに留意してください。

参照

[On Unload](#)

On Load Record

コード	呼び出し元	定義
40	フォーム	リスト更新中にレコードがロードされ、フィールドが編集モードに入った

説明

`On Load Record` イベントは、出力フォーム のコンテキストでのみ使用できます。このイベントはリスト更新中に、レコードがハイライトされ、フィールドが編集モードになったときに生成されます。

このイベントは、プロジェクトフォームでは選択できず、テーブルフォーム でのみ利用できます。

On Long Click

コード	呼び出し元	定義
39	ボタン	ボタンがクリックされ、特定の時間以上マウスボタンが押され続けている

説明

このイベントはボタンがクリックされ、一定時間以上マウスボタンが押され続けていると生成されます。理論上、このイベントが生成されるためのクリック保持時間は、システムの環境設定に設定されたダブルクリックの間隔最大時間に等しくなります。

このイベントは、次のボタンスタイルで生成することができます：

- [ツールバー](#)
- [ペベル](#)
- [角の丸いペベル](#)
- [OS Xグラデーション](#)
- [OS Xテクスチャー](#)
- [Office XP](#)
- [ヘルプ](#)
- [サークル](#)
- [カスタム](#)

このイベントは一般的に、ロングクリック時にポップアップメニューを表示するために使用します。ユーザーがロングクリックが有効になる時間前にマウスボタンを離すと、[On Clicked](#) イベントが（有効であれば）生成されます。

参照

[On Alternative Click](#)

On Losing focus

コード	呼び出し元	定義
14	4D View Pro エリア - 4D Write Pro エリア - ボタン - チェックボックス - コンボボックス - フォーム - 階層リスト - 入力 - リストボックス - リストボックス列 - プラグインエリア - 進捗インジケーター - ラジオボタン - ルーラー - スピナー - ステッパー - サブフォーム - Webエリア	フォームオブジェクトがフォーカスを失った

説明

`On Losing Focus` イベントや `On Getting Focus` イベントを使って、**フォーカス可** オブジェクトのフォーカスの変更を処理できます。

サブフォームオブジェクト の場合、このイベントがプロパティリスト中でチェックされていれば、サブフォームオブジェクトのメソッド内で生成されます。このイベントは、サブフォームのフォームメソッドに送信されます。これによってたとえば、フォーカスに応じてサブフォーム中のナビゲーションボタンの表示を管理できます。サブフォームオブジェクトは、それ自体がフォーカスを持つ点に留意してください。

On Menu Selected

コード	呼び出し元	定義
18	フォーム	連結メニューバーのメニュー項目が選択された

説明

`On Menu Selected` イベントは、フォームに関連付けられたメニューバーのコマンドが選択されると、フォームメソッドに送られます。その後、`Menu selected` ランゲージコマンドを呼び出して、選択されたメニューをテストすることができます。

メニューバーをフォームに関連付けるには、フォームのプロパティで設定します。フォームメニューバーのメニューは、アプリケーション環境でフォームが出力フォームとして表示されたときに、現在のメニューバーに追加されます。

On Mouse Enter

コード	呼び出し元	定義
35	4D Write Pro エリア - ボタン - ボタングリッド - チェックボックス - コンボボックス - ドロップダウンリスト - フォーム - 階層リスト - 入力 - リストボックス - ピクチャーボタン - ピクチャーポップアップメニュー - ブラグインエリア - 進捗インジケーター - ラジオボタン - ルーラー - スピナー - スプリッター - ステッパー - タブコントロール	マウスカーソルがオブジェクトの描画エリア内に入った

説明

このイベントは、マウスカーソルがフォームオブジェクトの描画エリアに入ったときに一度だけ発生します。

On Mouse Enter イベントは、*MouseX* および *MouseY* システム変数を更新します。

OBJECT SET VISIBLE コマンドの使用や、**表示状態** プロパティの設定によって非表示にされたオブジェクトでは、このイベントは生成されません。

コールスタック

On Mouse Enter イベントがフォームにおいてチェックされている場合、各フォームオブジェクトに対してイベントが生成されます。あるオブジェクトにおいてチェックされている場合は、そのオブジェクトに対してのみ生成されます。重なったオブジェクトがある場合はトップレベルから順に、イベントを処理することができる最初のオブジェクトによって生成されます。

参照

- [On Mouse Move](#)
- [On Mouse Leave](#)

On Mouse Leave

コード	呼び出し元	定義
36	4D Write Pro エリア - ボタン - ボタングリッド - チェックボックス - コンボボックス - ドロップダウンリスト - フォーム - 階層リスト - 入力 - リストボックス - ピクチャーボタン - ピクチャーポップアップメニュー - ブラグインエリア - 進捗インジケーター - ラジオボタン - ルーラー - スピナー - スプリッター - ステッパー - タブコントロール	マウスカーソルがオブジェクトの描画エリアから出た

説明

このイベントは、マウスカーソルがフォームオブジェクトの描画エリアから出たときに一度だけ発生します。

`On Mouse Leave` イベントは、`MouseX` および `MouseY` システム変数を更新します。

`OBJECT SET VISIBLE` コマンドの使用や、[表示状態](#) プロパティの設定によって非表示にされたオブジェクトでは、このイベントは生成されません。

コールスタック

`On Mouse Leave` イベントがフォームにおいてチェックされている場合、各フォームオブジェクトに対してイベントが生成されます。あるオブジェクトにおいてチェックされている場合は、そのオブジェクトに対してのみ生成されます。重なったオブジェクトがある場合はトップレベルから順に、イベントを処理することができる最初のオブジェクトによって生成されます。

参照

- [On Mouse Move](#)
- [On Mouse Leave](#)

On Mouse Move

コード	呼び出し元	定義
37	4D Write Pro エリア - ボタン - ボタングリッド - チェックボックス - コンボボックス - ドロップダウンリスト - フォーム - 階層リスト - 入力 - リストボックス - ピクチャーボタン - ピクチャーポップアップメニュー - ブラグインエリア - 進捗インジケーター - ラジオボタン - ルーラー - スピナー - スプリッター - ステッパー - タブコントロール	マウスカーソルがオブジェクトの描画エリア上で (最低1ピクセル) 動いたか、変更キー (Shift, Alt/Option, Shift Lock) が押された

説明

このイベントは次の場合に生成されます:

- マウスカーソルが 1ピクセル以上動いた
- または、変更キー (Shift, Alt/Option, Shift Lock) が押された これにより、ドラッグ & ドロップによるコピーや移動も管理できるようになります

イベントがオブジェクトに対してのみチェックされていた場合は、マウスカーソルがオブジェクトの描画エリア内にあった場合にのみイベントが生成されます。

`On Mouse Move` イベントは、`MouseX` および `MouseY` システム変数を更新します。

`OBJECT SET VISIBLE` コマンドの使用や、[表示状態](#) プロパティの設定によって非表示にされたオブジェクトでは、このイベントは生成されません。

コールスタック

`On Mouse Move` イベントがフォームにおいてチェックされている場合、各フォームオブジェクトに対してイベントが生成されます。あるオブジェクトにおいてチェックされている場合は、そのオブジェクトに対してのみ生成されます。重なったオブジェクトがある場合はトップレベルから順に、イベントを処理することができる最初のオブジェクトによって生成されます。

参照

- [On Mouse Enter](#)
- [On Mouse Leave](#)

On Mouse Up

コード	呼び出し元	定義
2	ピクチャー型の入力	ユーザーがピクチャーオブジェクト内にて左マウスボタンを離した

説明

ピクチャー型の入力オブジェクト内で、ドラッグ中に左マウスボタンをリリースしたときに `On Mouse Up` イベントが生成されます。このイベントはたとえば、SVGエリア内でユーザーがオブジェクトを移動、リサイズ、描画することを可能にしたい場合に有用です。

`On Mouse Up` イベントが生成されると、マウスボタンがリリースされたローカルの座標を取得することができます。座標は `MouseX` と `MouseY` システム変数に返されます。座標はピクセル単位で表現され、ピクチャーの左上隅が起点 (0,0) となります。

このイベントを使用する場合、フォームの "ステートマネージャー" が非同期の可能性がある場合 (つまり、クリック後に `On Mouse Up` イベントを受け取らなかった場合) を管理するために、`Is waiting mouse up` コマンドも使用する必要があります。これはたとえば、マウスボタンがリリースされる前にフォーム上にアラートダイアログボックスが表示された場合などです。`On Mouse Up` イベントのより詳細な情報と使用例については、`Is waiting mouse up` コマンドの詳細を参照ください。

ピクチャーオブジェクトの [ドラッグ有効](#)オプションがチェックされている場合、`On Mouse Up` イベントはいかなる場合も生成されません。

On Open Detail

コード	呼び出し元	定義
25	フォーム - リストボックス	出力フォームまたはリストボックスに関連付けられた詳細フォームが開かれようとしている

説明

`On Open Detail` イベントは次のコンテキストで利用できます:

- 出力フォーム: 出力フォームに関連づけられた詳細フォームにレコードが表示されようとしているときに生成されます。このイベントは、プロジェクトフォームでは選択できず、テーブルフォーム でのみ利用できます。
- [セレクション型](#) リストボックスに関連付けられた (まだ開かれていない) 詳細フォームにレコードが表示されようとしているときに生成されます。

Displayed line number

`Displayed line number` 4Dコマンドは、`On Open Detail` フォームイベントと連動します。このコマンドは、レコードのリストまたはリストボックスの行が画面に表示されるときに処理されている行の番号を返します。

On Open External Link

コード	呼び出し元	定義
52	Webエリア	外部URL がブラウザーで開かれた

説明

このイベントは、WA SET EXTERNAL LINKS FILTERS コマンドで設定されたフィルターにより、URL のロードが Webエリアによってブロックされ、URL がカレントのシステムブラウザーで開かれると生成されます。

WA Get last filtered URL コマンドコマンドを使用してブロックされた URL を知ることができます。

参照

[On URL Filtering](#)

On Outside Call

コード	呼び出し元	定義
10	フォーム	フォームが POST OUTSIDE CALL による呼び出しを受けた

説明

このイベントは、POST OUTSIDE CALL コマンドによって他のプロセスからフォームが呼び出されたときに生成されます。

On Outside Call イベントは、受信した入力フォームの入力コンテキストを変更します。特に、フィールドが編集されていた場合には、On Data Change イベントが生成されます。

On Page Change

コード	呼び出し元	定義
56	フォーム	フォーム中のカレントページが変更された

説明

このイベントは、フォームレベルでのみ利用できます（フォームメソッド内でのみ使用します）。このイベントは、（ `FORM GOTO PAGE` コマンドの呼び出しや標準ナビゲーションアクションに伴い）フォームのカレントページが変更されるたびに生成されます。

ページが完全にロードされた後（つまり、（ [Webエリア](#) を含む）すべてのオブジェクトが初期化された後）に呼び出されることに留意してください。

唯一の例外は 4D View Proエリアで、その場合は専用の [On VP Ready](#) イベントを呼び出す必要があります。

`On Page Change` イベントは、すべてのオブジェクトが初期化済みの状態で実行する必要のあるコードがあるときに有用です。また、フォームのページ 1 がロードされたときにすべてのコードを実行するのではなく、特定のページが開かれたときにのみ（検索などの）コードを実行するようにして、アプリケーションを最適化できます。ユーザーがそのページを開かなければ、コードは実行されません。

On Plug in Area

コード	呼び出し元	定義
19	フォーム - プラグインエリア	外部オブジェクトのオブジェクトメソッドの実行がリクエストされた

説明

このイベントは、関連するオブジェクトメソッドの実行をプラグインがフォームエリアに要求したときに発生します。

On Printing Break

コード	呼び出し元	定義
6	4D Write Pro エリア - ボタン - ボタングリッド - チェックボックス - コンボボックス - ドロップダウンリスト - フォーム - 階層リスト - 入力 - ピクチャーボタン - ピクチャーポップアップメニュー - プラグインエリア - 進捗インジケーター - ラジオボタン - ルーラー - スピナー - スプリッター - ステッパー - タブコントロール	フォームのブレークエリアが印刷されようとするたびに発生し、これによってブレーク値を評価できるようになります。

説明

`On Printing Break` イベントは、出力フォーム のコンテキストでのみ使用できます。このイベントは、出力フォームのブレークエリアが印刷されようとするとたびに発生し、これによってブレーク値を評価できるようになります。

このイベントは通常、`Subtotal` コマンドの呼び出し後に発生します。

このイベントは、プロジェクトフォームでは選択できず、テーブルフォーム でのみ利用できます。

On Printing Detail

コード	呼び出し元	定義
23	4D Write Pro エリア - ボタン - ボタングリッド - チェックボックス - コンボボックス - ドロップダウンリスト - フォーム - 階層リスト - 入力 - ピクチャーボタン - ピクチャーポップアップメニュー - プラグインエリア - 進捗インジケーター - ラジオボタン - ルーラー - スピナー - スプリッター - ステッパー - タブコントロール	フォームの詳細エリアが印刷されようとしている

説明

`On Printing Detail` イベントは、出力フォーム のコンテキストでのみ使用できます。このイベントは、`Print form` コマンドの呼び出しの後など、出力フォームの詳細エリアが印刷されようとしているときに発生します。

`Print form` コマンドは `On Printing Detail` イベントを 1つだけ、フォームメソッドに対して生成します。

このイベントは、プロジェクトフォームでは選択できず、テーブルフォーム でのみ利用できます。

On Printing Footer

コード	呼び出し元	定義
7	4D Write Pro エリア - ボタン - ボタングリッド - チェックボックス - コンボボックス - ドロップダウンリスト - フォーム - 階層リスト - 入力 - ピクチャーボタン - ピクチャーポップアップメニュー - プラグインエリア - 進捗インジケーター - ラジオボタン - ルーラー - スピナー - スプリッター - ステッパー - タブコントロール	フォームのフッターエリアが印刷されようとしている

説明

`On Printing Footer` イベントは、出力フォーム のコンテキストでのみ使用できます。このイベントは、出力フォームのフッターエリアが印刷されようとするときに発生し、これによってフッターバリューを評価できるようになります。

このイベントは、`PRINT SELECTION` コマンドのコンテキストで発生することが可能です。

このイベントは、プロジェクトフォームでは選択できず、テーブルフォーム でのみ利用できます。

On Resize

コード	呼び出し元	定義
29	フォーム	フォームウィンドウまたはサブフォームオブジェクトがリサイズされた（後者の場合は、サブフォームのフォームメソッドにおいてイベントが生成されます）

説明

このイベントは次の場合に生成されます:

- フォームのウィンドウがリサイズされた
- サブフォームのコンテキストでは、親フォームにおいてサブフォームオブジェクトのサイズが変更された場合。この場合、イベントはサブフォームのフォームメソッドが受け取ります。

On Row Moved

コード	呼び出し元	定義
34	配列型リストボックス - リストボックス列	リストボックスの行がユーザーのドラッグ & ドロップで移動された

説明

このイベントは、ユーザーのドラッグ & ドロップ ([許可されていれば](#)) で、リストボックス ([配列型のみ](#)) の行が移動されたときに生成されます。ただし、元の場所にドロップされた場合には生成されません。

`LISTBOX MOVED ROW NUMBER` コマンドは行の新しい位置を返します。

On Row Resize

コード	呼び出し元	定義
60	4D View Pro エリア	行の高さがユーザーのマウスによって変更された

説明

このイベントは、4D View Pro ドキュメント内で行の高さがユーザーによって変更されたときに生成されます。このコンテキストにおいて、 FORM Event コマンドによって返される イベントオブジェクト には以下のプロパティが含まれています：

プロパティ	タイプ	説明
code	倍長整数	60
description	テキスト	"On Row Resize"
objectName	テキスト	4D View Pro エリア名
sheetName	テキスト	イベントが発生したシート名
range	object	高さが変更された行のセルレンジ
headers	boolean	カラムヘッダー行（最初の行）がリサイズされた場合には true、それ以外の場合には false

例題

```
If(FORM Event.code=On Row Resize)
    VP SET CELL STYLE(FORM Event.range;New object("vAlign";vk vertical align top))
End if
```

On Scroll

コード	呼び出し元	定義
59	ピクチャー型の入力 - リストボックス	マウスやキーボードを使用して、ユーザーがピクチャーオブジェクトやリストボックスの内容をスクロールした。

説明

このイベントは、ピクチャー入力またはリストボックスのコンテキストで生成されます。

このイベントは、スクロールアクションに関連した他のすべてのユーザーイベント ([On Clicked](#)、[On After Keystroke](#)、等) の後にトリガーされます。このイベントは、オブジェクトメソッドの中でのみ生成されます (フォームメソッドでは生成されません)。

このイベントは、ユーザーのアクションの結果としてスクロールが発生した場合にのみ生成されます: スクロールバー/カーソルの使用、マウスホイールまたは[キーボード](#)の使用、等です。`OBJECT SET SCROLL POSITION` コマンド使用の結果スクロールした場合には生成されません。

ピクチャー入力

このイベントは、ユーザーがピクチャー入力 (フィールドや変数) 内のピクチャーをスクロールすると生成されます。ピクチャーエリアのサイズがピクチャーよりも小さく、かつ表示テーマの[ピクチャーフォーマット](#)プロパティが "トランケート (中央合わせなし)" に設定されている場合のみスクロールが可能です。

リストボックス

このイベントは、ユーザーがリストボックスの行や列をスクロールすると生成されます。

On Selection Change

コード	呼び出し元	定義
31	4D View Pro エリア - 4D Write Pro エリア - フォーム - 階層リスト - 入力 - リストボックス	オブジェクト内で選択が変更された

説明

このイベントは様々なコンテキストで発生します。

4D View Pro

現在の行や列の選択が変更された。このコンテキストにおいて、FORM Event コマンドによって返される イベントオブジェクト には以下のプロパティが含まれています:

プロパティ	タイプ	説明
code	倍長整数	31
description	テキスト	"On Selection Change"
objectName	テキスト	4D View Pro エリア名
sheetName	テキスト	イベントが発生したシート名
oldSelections	object	変更前のセルレンジ
newSelections	object	変更後のセルレンジ

例題

```
If(FORM Event.code=On Selection Change)
    VP SET CELL STYLE(FORM Event.oldSelections;New object("backColor";Null))
    VP SET CELL STYLE(FORM Event.newSelections;New object("backColor";"red"))
End if
```

リストフォーム

リストフォームにおいて、カレントレコードあるいはカレントセレクションの行選択が変更された。

階層リスト

階層リスト中の選択がクリックやキーストロークなどで変更された。

入力 & 4D Write Pro

クリックやキーストロークにより、選択されたテキストやカーソルの位置がエリア内で変更された。

リストボックス

このイベントは、リストボックス内で現在の行や列の選択が変更されるたびに生成されます。

On Timer

コード	呼び出し元	定義
27	フォーム	SET TIMER コマンドで設定した時間（ティック数）が経過した

説明

このイベントは、事前に SET TIMER コマンドが使用された場合にのみ生成されます。

On Timer フォームイベントプロパティが選択されると、フォームメソッドのみがイベントを受け取ります。オブジェクトメソッドは呼び出されません。

On Unload

コード	呼び出し元	定義
24	4D View Pro エリア - 4D Write Pro エリア - ボタン - ボタングリッド - チェックボックス - コンボボックス - ドロップダウンリスト - フォーム - 階層リスト - 入力 - リストボックス - リストボックス列 - ピクチャーボタン - ピクチャーポップアップメニュー - プラグインエリア - 進捗インジケーター - ラジオボタン - ルーラー - スピナー - スプリッター - ステッパー - サブフォーム - タブコントロール - Webエリア	フォームを閉じて解放しようとしている

説明

このイベントは、フォームを閉じて解放しようとしているときに生成されます。

`On Unload` オブジェクトイベントプロパティが選択されている、フォームの全ページの全オブジェクトのオブジェクトメソッドが呼び出されます。その後、`On Unload` フォームイベントプロパティが選択されていれば、フォームメソッドが呼び出されます。

オブジェクトの `On Load` と `On Unload` イベントが生成されるには、オブジェクトとオブジェクトが属するフォームの両方で有効にされていなければなりません。オブジェクトのみでイベントが有効になっている場合、イベントは生成されません。これら 2つのイベントはフォームレベルでも有効にされていなければなりません。

サブフォーム

`On Unload` イベントは、サブフォームを閉じる際に生成されます。これらのイベントは親フォームレベルでも有効にされていなければなりません。このイベントは、親フォームのイベントよりも前に生成される点に留意してください。また、フォームイベント動作の原則に従い、サブフォームが 0 もしくは 1 以外のページに配置されている場合、このイベントはページが閉じられるときに生成され、フォームが閉じられるときではないことに留意してください。

参照

[On Load](#)

On URL Filtering

コード	呼び出し元	定義
51	Webエリア	Webエリアが URL をブロックした

説明

このイベントは、`WA SET URL FILTERS` コマンドで設定されたフィルターにより、Webエリアによって URL のロードがブロックされると生成されます。

`WA Get last filtered URL` コマンドコマンドを使用してブロックされた URL を知ることができます。

参照

[On Open External Link](#)

On URL Loading Error

コード	呼び出し元	定義
50	Webエリア	URL をロード中にエラーが発生した

説明

このイベントは、URL のロード中にエラーを検出すると生成されます。

`WA GET LAST URL ERROR` コマンドを使用して、エラーに関する情報を取得できます。

参照

[On Open External Link](#)

On URL Resource Loading

コード	呼び出し元	定義
48	Webエリア	新しいリソースが Webエリアにロードされた

説明

このイベントは、現在の Webページに (ピクチャやフレームなどの) 新しいリソースをロードするたびに生成されます。

Webエリアに関連付けられた [進捗状況変数](#) 変数を使用してロード状況を知ることができます。

参照

[On Open External Link](#)

On Validate

コード	呼び出し元	定義
3	4D Write Pro エリア - ボタン - ボタングリッド - チェックボックス - コンボボックス - ドロップダウンリスト - フォーム - 階層リスト - 入力 - リストボックス - リストボックス列 - ピクチャーボタン - ピクチャーポップアップメニュー - プラグインエリア - 進捗インジケーター - ラジオボタン - ルーラー - スピナー - スプリッター - ステッパー - サブフォーム - タブコントロール	レコードのデータ入力が受け入れられた

説明

このイベントは、`SAVE RECORD` コマンドコールや `accept` [標準アクションの後](#) など、レコードデータの入力が受け入れられたときにトリガーされます。

サブフォーム

`On Validate` イベントは、サブフォーム中でデータの受け入れをおこなう際に発生します。

On VP Range Changed

コード	呼び出し元	定義
61	4D View Pro エリア	4D View Pro のセルレンジが変更された（例：フォーミュラの計算、値がセルから削除された、など）

説明

このイベントは、4D View Pro ドキュメント内でセルレンジが変更されたときに生成されます。

FORM Event によって返されるオブジェクトには以下のプロパティが格納されます：

プロパティ	タイプ	説明
objectName	テキスト	4D View Pro エリア名
code	倍長整数	On VP Range Changed
description	テキスト	"On VP Range Changed"
sheetName	テキスト	イベントが発生したシート名
range	object	変化したセルレンジ
changedCells	object	変化したセルのみを格納したレンジ。レンジが組み合わされたものである可能性もあります。
action	テキスト	イベント生成した操作のタイプ: <ul style="list-style-type: none">● "clear" - レンジの値をクリア操作● "dragDrop" - ドラッグドロップ操作● "dragFill" - ドラッグによるフィル操作● "evaluateFormula" - 特定のセルレンジにフォーミュラを設定した● "paste" - ペースト操作● "setArrayFormula" - 特定のセルレンジにフォーミュラを設定した● "sort" - セルのレンジを並べ替えた

[On After Edit](#) も参照ください。

On VP Ready

コード	呼び出し元	定義
9	4D View Pro エリア	4D View Pro エリアのロードが完了した

説明

このイベントは、4D View Pro エリアのロードが終わった時に生成されます。

このイベントは、エリアの初期化コードを書く際に必要になります。4D View Pro エリアを初期化するためのコード（値の読み込みなど）は当該エリアの `On VP Ready` フォームイベント内に書く必要があります。このフォームイベントは、エリアの読み込みが完了したときに一回生成されます。同イベントの利用によって、有効なコンテキスト内でコードが実行されることを確実にできます。`On VP Ready` フォームイベントが生成される前に 4D View Pro コマンドを呼び出してしまうと、エラーが返されます。

4D View Pro エリアは非同期的に 4D フォーム上に読み込まれます。したがって、標準の `On load` フォームイベントは 4D View Pro エリアの読み込み完了を保証せず、4D View Pro の初期化コードに利用することはできません。`On VP Ready` は常に `On load` の後に生成されます。

On Window Opening Denied

コード	呼び出し元	定義
53	Webエリア	ポップアップウィンドウがブロックされた

▶ 覆歴

説明

このイベントは、Webエリアによりポップアップウィンドウがブロックされると生成されます。4D Webエリアはポップアップウィンドウを許可しません。

WA Get last filtered URL コマンドコマンドを使用してブロックされた URL を知ることができます。

このイベントは、Webエリア（埋め込みおよび Windowsシステム エンジン）で ドラッグ & ドロップ オプションが有効になっている場合に、ドロップ操作がおこなわれたときにも発生します。次を呼び出すことで、ドロップを受け入れることができます：

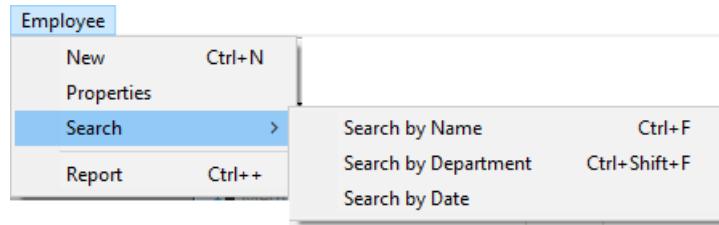
```
// Webエリアオブジェクトメソッド
If (FORM Event.code=On Window Opening Denied)
    WA OPEN URL(*; "WebArea"; WA Get last filtered URL(*; "WebArea"))
    // または UrlVariable:=WA Get last filtered URL(*; "WebArea")
    // (UrlVariable は Webエリアに関連づけられた URL変数)
End if
```

参照

On Open External Link

概要

4D アプリケーション用にカスタムメニューを作成できます。デスクトップアプリケーションではプルダウン形式のメニューが標準機能であるため、メニューを追加することでアプリケーションがより使いやすくなりユーザーに親しみやすいものになるでしょう。



メニュー バーとは、スクリーン上にまとめて表示されるメニューのグループです。メニュー バー上の各 メニューはメニュー コマンドを持ちます。またメニュー コマンドは階層メニューと呼ばれるサブメニューを持つこともできます。メニュー やサブメニュー コマンドをユーザーが選択すると、プロジェクト メソッドまたは標準アクションが呼び出されます。

各アプリケーションに対し、異なるメニュー バーを複数作成することもできます。たとえば、あるメニュー バーには標準的なデータベース処理用のメニューを納め、別のメニュー バーはレポート作成時にのみアクティブにすることができます。また別のメニュー バーには、レコード入力用のメニュー コマンドを含むメニューを格納することも可能です。入力フォームと一緒に表示されるメニュー バーには同じメニューを格納しながらも、データ入力中は不要になるメニュー コマンドを選択不可にすることができます。

あるメニューを複数のメニュー バーで使用したり、どのメニュー バーにも割り当てずにプログラムからのみ管理することもできます（独立メニューと呼びます）。

メニューを設計する際には以下の 2つのルールを覚えておいてください：

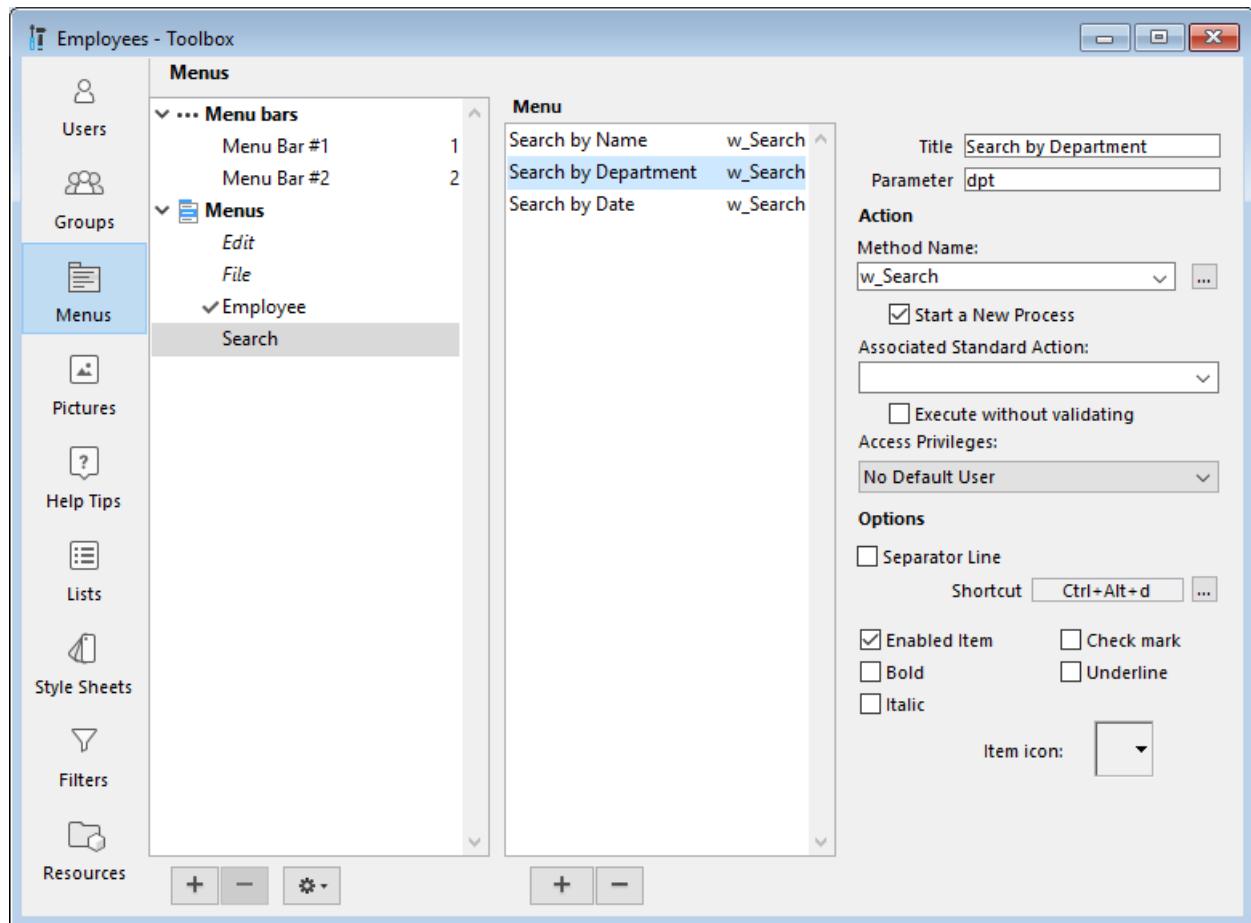
- メニューに適している機能に対しメニューを使用する：メニュー コマンドは、レコードの追加や検索、レポートの印刷のような作業を実行しなければなりません。
- メニュー コマンドを機能別にまとめる：たとえば、レポートの印刷をおこなうメニュー コマンドはすべて同じメニュー 内に置くべきです。また別の例として、特定のテーブルに関するすべての操作を 1つのメニューに納めてもよいでしょう。

メニュー やメニュー バーを作成するには以下のいずれかを使用します：

- ツールボックスのメニュー エディター
- "メニュー" テーマのランゲージコマンド
- 上 2つの組み合わせ

メニュー エディター

ツールボックスのメニュー ページを開くと、メニュー エディターにアクセスできます。



メニューおよびメニューバーは両方ともエディター左の階層リスト中に表示されます。各メニューは、メニューバーあるいは別のメニューに添付できます。後者の場合、そのメニューはサブメニューとなります。

4D はメニューバーに連番を割り当てます。メニューバー#1 が一番上に表示されます。メニューバーの名前を変更することができますが、番号は変更できません。この番号はランゲージコマンドで使用されます。

メニューとメニューバーの作成

メニューおよびメニューバーを作成するには次の 2つの方法があります:

- 4Dツールボックスウンドウのメニューエディターを使用する。この場合、メニューとメニューバーはアプリケーションのストラクチャーに保存されます。
- "メニュー" テーマのランゲージコマンドを使用して動的におこなう。この場合、メニューとメニューバーは保存されず、メモリ内にのみ存在します。

両方の機能を組み合わせて、メモリ内のメニューを定義するのに、ストラクチャーに作成したメニューをテンプレートとして使うこともできます。

デフォルトメニューバー

カスタムアプリケーションには、少なくとも 1つのメニューを持つ 1つのメニューバーが必要です。新規にプロジェクトを作成すると、4D は自動でデフォルトメニューバー (メニューバー#1) を作成します。このデフォルトメニューバーには、標準のメニューとデザインモードに入るためのコマンドが用意されています。

このメニューが用意されているため、ユーザーはプロジェクトを起動するとすぐにアプリケーションモードを使用できます。実行 メニューから アプリケーションモード コマンドを選択すると、自動でメニューバー#1 が呼び出されます。

デフォルトメニューバーには 3つメニューがあります:

- ファイル: このメニューには 終了 コマンドだけが含まれています。このコマンドには *quit* 標準アクションが割り当てられていて、選択されるとアプリケーションが終了します。
- 編集: 編集メニューは標準であり、内容の変更が可能です。編集メニューのコマンド (コピーやペーストなど) は標準アクションで指定できます。
- モード: モードメニューにはデフォルトで、アプリケーションモードを終了するための デザインモードに戻る コマンドが含まれます。

メニュータイトルはハードコードされたテキストではなく、xliff 参照を使用しています。この点については [タイトルプロパティ](#) を参照してください。

このメニューbaruを必要に応じて変更したり、新しく追加したりできます。

メニューの作成

メニューエディターを使用する

1. 作成する対象 (メニューbaruまたはメニュー) を選択し、エリアの下にある追加ボタン をクリックします。またはリストのコンテキストメニューまたはリストの下にあるオプションメニューから 新規メニューbaru作成 あるいは 新規メニュー作成 を選択します。メニューbaruを作成した場合は、新しいメニューbaruがリスト中に追加され、デフォルトメニュー (ファイルと編集) があらかじめ添付されています。
2. (任意) メニューbaru/メニューの名前の上でダブルクリックすると、名前を編集できるモードになり、名前を変更することができます。またはウインドウ右の "タイトル" エリアに名前を入力します。メニューbaru名はユニークでなければなりません。名前には 31文字までの文字列を指定できます。メニューのタイトルには文字列リテラルのほかに、参照も使用できます ([タイトルプロパティ](#) の説明を参照ください)。

4Dランゲージを使用する

`Create menu` コマンドを使って、新規メニューbaruまたはメニュー参照 (`MenuRef`) をメモリ上に作成します。

メニューが `MenuRef` 参照を使用して処理される場合、メニューとメニューbaruの間に違いはありません。両方とも項目のリストから構成されます。それらの利用方法のみが異なります。メニューbaruの各項目は、それ自身が 1つのメニューであり、項目から構成されています。

`Create menu` で空のメニューを作成した場合には、`APPEND MENU ITEM` または `INSERT MENU ITEM` コマンドによって項目を追加していきます。また、同コマンドのソースメニューとして、メニューエディターで定義されたメニューを指定した場合には、そのコピーが新しいメニューとして作成されます。

項目の追加

各メニューには、メニューがクリックされたときにドロップダウン表示されるメニュー項目を作成しなければなりません。項目を追加してメソッドや標準アクションを割り当てたり、他のメニューをサブメニューとして添付したりできます。

メニューエディターを使用する

メニュー項目を追加するには:

- ソースメニューリスト中で、項目を追加するメニューを選択します。メニューが既に項目を持っていれば、それが中央のリストに表示されます。新しい項目を挿入するには、その上にくる項目を選択します。ドラッグ & ドロップ操作で、後から順番を変更することも可能です。
- メニューエディターのオプションメニュー、またはエディターのコンテキストメニュー（中央のリスト内で右クリック）から メニューバー/メニュー "メニュー名" に項目を追加を選択します。または
中央のリストの下にある追加ボタン をクリックします。項目が追加され、デフォルト名 "項目 X" が割り当てられます (X は項目の番号)。
- 項目名の上でダブルクリックすると、名前を編集できるモードになり、名前を変更することができます。または
ウインドウ右の "タイトル" エリアに名前を入力します。名前には 31 文字までの文字列を指定できます。メニューのタイトルには文字列リテラルのほかに、参照も使用できます（後述参照）。

4Dランゲージを使用する

既存のメニュー参照にメニュー項目を挿入するには `INSERT MENU ITEM` または `APPEND MENU ITEM` コマンドを使います。

メニューと項目の削除

メニューと項目の削除

メニューと項目の削除は、メニューと項目を一つでも削除できます。各メニューとメニューは 1 つの参照しか持たない点に留意してください。あるメニューが、複数のメニューとメニューに添付されていた場合、そのメニューに対しておこなわれた変更や削除はこのメニューのすべての他のオカレンスに対しても有効となります。メニューを削除すると、参照だけが削除されます。特定のメニューへの最後の参照を削除しようとすると、4D はアラートを表示します。

メニューとメニューと項目を削除するには:

- 削除する項目を選択し、リストの下にある削除ボタン をクリックします。
- コンテキストメニューまたはエディターのオプションメニューから ~ を削除 コマンドを選択します。

メニュー #1 を削除することはできません。

4Dランゲージを使用する

`DELETE MENU ITEM` コマンドを使ってメニュー参照から項目を削除します。メニュー参照をメモリからアンロードするには `RELEASE MENU` コマンドを使います。

メニューと項目の添付

メニューを作成したら、それをメニューと別のメニューに（サブメニューとして）添付できます。

サブメニューは、テーマに基づき機能をグループ化する目的で使用されます。サブメニューとその項目は、メニューと同じ属性（アクション、メソッド、ショートカット、アイコン等）を持つことができます。サブメニューの項目は元の特性やプロパティを保持し、サブメニューの動作は標準のメニューと同じです。

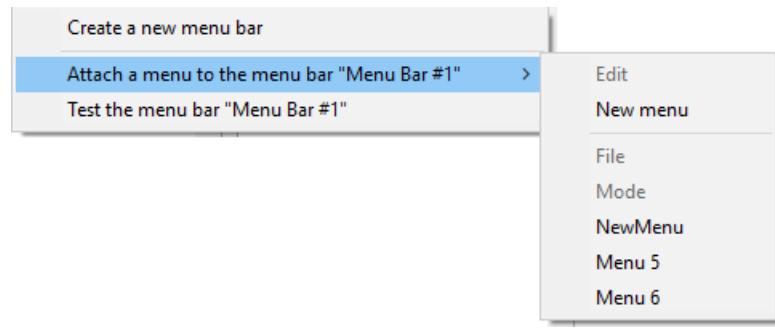
サブメニューのサブメニューを作成することができ、階層化に制限はありません。しかし、インターフェース標準に沿うには、2 レベルを超えるサブメニューは推奨されません。

ランタイムにおいてプログラミングによりメニューを変更した場合、そのメニューが添付されているすべてのインスタンスに変更が反映されます。

メニューと項目の添付

各メニューは、メニューとあるいは別のメニューに添付できます。

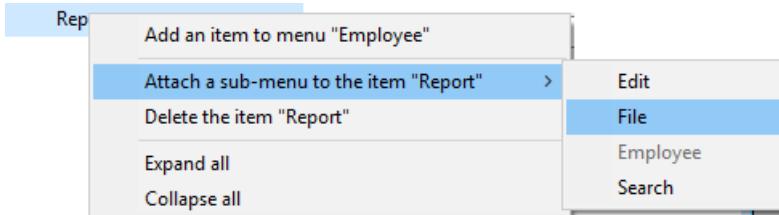
- メニューとメニューを添付するには: メニューバーを右クリックし、メニュー "メニュー名" にメニューを添付 > を選択、そしてサブメニューから



添付するメニューを選択します:

リストの下にあるオプションメニューをクリックする方法もあります。

- メニューに他のメニューを添付するには: 左のリスト中でメニューを選択し、次に中央リストのメニュー項目上で右クリックし、項目 "項目名" にサブメニューを添付 > を選択、そしてサブメニューとして使用するメニューを選択します:



メニュー項目を選択してから、リストの下にあるオプションメ

ニューをクリックする方法もあります。添付されたメニューはサブメニューとなります。項目のタイトルは保持されますが（元のサブメニュー名は無視されます）、このタイトルを変更することができます。

メニューの分離

メニューからメニューを、あるいはメニューからサブメニューを分離できます。分離されたメニューは、メニューから利用できなくなります。しかしメニューリストには残されます。

メニューを分離するには、中央リスト内の分離したいメニュー上で右クリックし、メニュー "メニュー名" をメニュー "メニュー名" から分離 または 項目 "項目名" のサブメニューを分離 を選択します。

4Dランゲージを使用する

4Dランゲージにおいてはメニューとメニューの違いはないため、メニューおよびサブメニューの添付は同じ手順でおこないます: APPEND MENU ITEM コマンドの subMenu パラメーターを指定して、メニューからメニューを添付します。

メニュー プロパティ

アクション、フォントスタイルや区切り線、キーボードショートカット、アイコンなど様々なメニュー項目プロパティを設定できます。

タイトル

タイトル プロパティには、アプリケーションインターフェースに表示されるメニュー/メニュー項目のラベルを指定します。

メニュー エディターを使って、テキストリテラルを直接、ラベルとして入力することができます。または、変数参照、xliff 参照を使用することもできます。これによりアプリケーションの翻訳が容易になります。次の参照タイプを使用できます：

- :xliff:MyLabel という形の XLIFFリソース参照。XLIFF参照についての詳細は、4D デザインリファレンスの [XLIFF アーキテクチャ](#) の章を参照ください。
- :<>vlang,3 という形のインターポセス変数名と、それに続く数値。この変数の内容を変更すると、メニューが表示される際にラベルも変更されます。この場合、ラベルは XLIFFリソースを呼び出します。<>vlang 変数に含まれる値は group 要素の id 属性値に対応します。二つ目の値(例では3)は trans-unit 要素の id 属性の値を指定します。

4D ランゲージを使う場合は、`APPEND MENU ITEM`、`INSERT MENU ITEM`、および `SET MENU ITEM` コマンドの `itemText` パラメーターでタイトル プロパティを設定します。

制御文字の使用

メニュー タイトルに制御文字(メタ文字)を直接使用し、メニューのプロパティをいくつか設定することができます。たとえば、メニュー タイトルに "/G" という文字を入れると、キーボードショートカットである Ctrl+G (Windows) または Command+G (macOS) を割り当てるすることができます。

制御文字は、表示されるメニュー ラベルには含まれません。したがって、制御文字として利用しない場合は、これらの文字の並びをタイトルに使用しないことが推奨されます。制御文字には次のようなものがあります：

文字	説明	効果
(開く括弧	項目を無効化
<B	小なりB	太字
<I	小なりI	イタリック
<U	小なりU	下線
!文字	エクスクラーメーションマーク+文字	文字をチェックマークとして追加 (macOS); チェックマークを追加 (Windows)
/文字	スラッシュ+文字	文字をショートカットとして追加

引数

各メニュー項目にカスタム引数を関連付けることができます。メニュー項目の引数は、その内容を自由に設定できる文字列です。メニュー エディターで設定するほかに、`SET MENU ITEM PARAMETER` コマンドを使う方法もあります。

メニュー項目の引数は、とくに `Dynamic pop up menu`、`Get menu item parameter`、そして `Get selected menu item parameter` コマンドを使用する際など、メニューをプログラムで管理するのに便利です。

アクション

メニューにはプロジェクト メソッドや標準アクションを割り当てるすることができます。メニュー項目が選択されると、4D は割り当てられた標準アクションまたはプロジェクト メソッドを実行します。例えば、月次報告書 メニューコマンドを設定し、財務データを格納したテーブルをもとに月次報告書を作成するプロジェクト メソッドを呼び出すことができます。カット メニューコマンドは、`cut` 標準アクションを呼び出して、選択項目をクリップボードへ移動し、それを前面にある ウィンドウから消去します。

標準アクションやメソッドをメニューに割り当てていない場合、そのメニューを選択すると、4D はアプリケーション環境からデザインモードに戻ります。デザインモードに移行できない場合は、4D を終了します。

標準アクションは、システム機能に関連した操作（コピー、終了、等）や、データベースに関連した操作（レコード追加、全選択、等）を実行するのに使用します。

標準アクションとプロジェクトメソッドの両方をメニューに割り当てるこども可能です。この場合、標準アクションが実行されることはありません。しかし、4D はこのアクションを使用し、状況に合わせてメニュー命令を使用可／使用不可に設定します。メニューが使用不可の場合、割り当てられたプロジェクトメソッドは実行されません。

求める結果の種類によって、標準アクションまたはプロジェクトメソッドのいずれを割り当てるかを選択します。原則として、標準アクションは最適化された方法で実行される（コンテキストに応じたメニューの有効/無効の自動切り替え）ため、できるだけこちらを選ぶ方が良いでしょう。

プロジェクトメソッドまたは標準アクションの割り当て

メニューエディターにて、標準アクション/プロジェクトメソッドをメニューに割り当てるすることができます：

- メソッド名：既存のプロジェクトメソッドをコンボボックスで選択します。プロジェクトメソッドがまだ存在しない場合、"メソッド名" コンボボックスにメソッド名を入力し、 [...] ボタンをクリックします。すると、4D はメソッド作成ダイアログボックスを表示し、メソッドエディターを開きます。
- 標準アクション：割り当てたいアクションを "標準アクション" コンボボックスから選択するか、記述します。サポートされているアクションと引数（任意）であれば、エリア内に入力することができます。標準アクションの一覧については、デザインリファレンスの [標準アクション](#) を参照してください。
macOS に関する注記：macOS の場合、プラットフォームインターフェース標準に合わせるために、*quit*（終了）アクションが割り当てられたカスタムメニュー命令は自動でアプリケーションメニュー内に置かれます。

4Dランゲージで割り当てをおこなう場合、プロジェクトメソッドには `SET MENU ITEM METHOD` コマンド、標準アクションには `SET MENU ITEM PROPERTY` コマンドを使います。

新規プロセスで開始

メソッドを割り当てたメニューの場合、新規プロセスで開始 オプションが利用可能です。このオプションは、メニューエディターのチェックボックスによって設定するほかに、`SET MENU ITEM PROPERTY` コマンドに *property* 引数を渡して設定することもできます。

新規プロセスで開始 チェックボックスを選択した場合、4D はそのメニュー命令が選択されると新しいプロセスを作成します。通常、メニュー命令に割り当てたメソッドは、明示的にプログラムから新規プロセスを作成しない限り、カレントプロセスで実行されます。新規プロセスで開始 チェックボックスを選択すると、新規プロセスを簡単に開始することができます。このチェックボックスを選択した場合は、このメニュー命令を選択すると新規プロセスが作成されます。

プロセスリストにおいて、4D は "ML_プロセス番号" というフォーマットのデフォルト名を新規プロセスに割り当てます。このように、メニューから開始されたプロセスの名前は、接頭辞 "ML_" とプロセス番号を組み合わせて設定されます。

イベントを発生させない

標準アクションを割り当てたメニューの場合、イベントを発生させない オプションがメニューエディター内で利用可能です。

このオプションを選択すると、4D は関連アクションを実行する前に、カーソルが置かれているフィールドの "確定 (バリデート)" をおこないません。このオプションは主に 編集 メニュー命令に使用されます。デフォルトでは、4D はフィールド内容を処理し、"確定" してから、標準アクションを実行します（メニュー命令やショートカットを使用）。これにより、`On Data Change` フォームイベントが発生します。しかし、コピー & ペーストタイプのコマンドの場合、コマンドを呼び出すたびに `On Data Change` フォームイベントが予想外に生成されてしまい、処理に差し支える可能性があります。その場合は、イベントを発生させない オプションを選択するのが有効です。

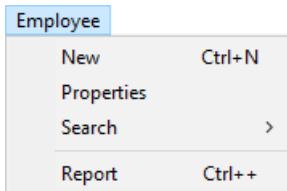
アクセス権

このメニューエディターのオプションを使って、メニュー命令にアクセスグループを設定することができます。4Dリモートアプリケーションで接続する場合、当該アクセスグループのユーザーのみがこのメニューを使うことができます（ユーザー & グループ参照）。

オプション

区切り線

メニュー内のメニュー命令グループは区切り線を使って分割できます。この表示方法は、機能ごとにメニュー命令をグループ化するのに便利です。



区切り線を追加するには、専用のメニュー命令を作成します。

メニューエディターでは、メニューのタイトルエリアにテキストを入力する代わりに、区切り線 オプションを選択します。すると、カレントメニューバーのエリアに線が表示されます。このオプションが選択されると、ほかのプロパティは無効になります。注: macOS ではメニュー項目タイトルの一文字目を "-" にすると、その行が区切り線になります。

4Dランゲージを使う場合は、APPEND MENU ITEM、INSERT MENU ITEM、または SET MENU ITEM コマンドの itemText パラメーターに `-` あるいは `(-)` を受け渡します。

ショートカット

メニュー命令にはショートカットを割り当てることができます。メニュー命令にキーボードショートカットが割り当てられると、メニューを開いたときにそれがメニュー命令の右に表示されます。たとえば、編集メニューのコピー命令の右に "Ctrl+C" (Windows) または "Command+C" (macOS) と表示されます。

ショートカットには Shift や Alt (Windows) または Option (macOS) キーを追加できます。これにより使用できるショートカットの数を増やすことができます。以下のタイプのショートカットを定義できます:

- Windows:
 - Ctrl+文字
 - Ctrl+Shift+文字
 - Ctrl+Alt+文字
 - Ctrl+Shift+Alt+文字
- macOS:
 - Command+文字
 - Command+Shift+文字
 - Command+Option+文字
 - Command+Shift+Option+文字

標準アクションに割り当てられたデフォルトのキーボードショートカットは変更しないことをお勧めします。

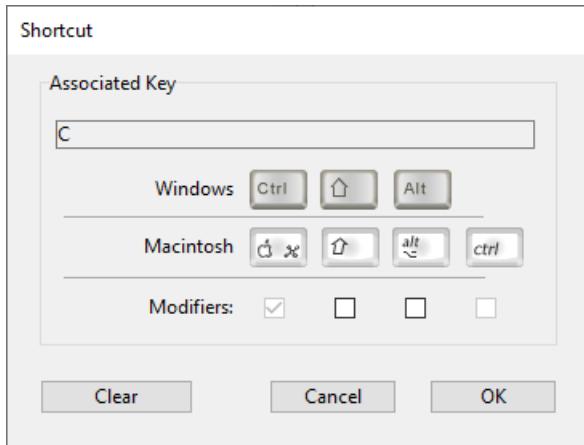
ファイル や 編集 メニュー、および 4D のメニュー命令に予約されている標準メニューのショートカットを除き、すべての英数字をキーボードショートカット文字として使用できます。

予約されている組み合わせは以下の通りです:

キー (Windows)	キー (macOS)	演算子
Ctrl+C	Command+C	コピー
Ctrl+Q	Command+Q	終了
Ctrl+V	Command+V	ペースト
Ctrl+X	Command+X	カット
Ctrl+Z	Command+Z	取り消し
Ctrl+. (ピリオド)	Command+. (ピリオド)	実行停止

メニューエディターでキーボードショートカットを割り当てるには:

キーボードショートカットを割り当てるメニュー項目を選択します。"ショートカット" 入力エリアの [...] ボタンをクリックします。以下のウインドウが表示されます:



文字を入力し、(必要であれば) Shift そして Alt (Option) チェックボックスを選択します。指定する組み合わせのキーを押すと、押したキーがウィンドウに反映されます (このときには Ctrl/Command キーは押しません)。

Ctrl/Command キーの選択を解除することはできません。このキーは必須です。内容を消去するには **クリア** をクリックします。 **OK** をクリックすると、内容を確定してウィンドウを閉じます。指定したショートカットが "ショートカット" 入力エリアに表示されます:

4D ランゲージでキーボードショートカットを割り当てるには、**SET ITEM SHORTCUT** コマンドを使います。

アクティブオブジェクトにも、キーボードショートカットを割り当てることができます。Ctrl/Command キーの割り当てが衝突した場合、アクティブオブジェクトが優先されます。

選択可

メニューエディターにて、メニュー項目を有効として表示するか無効として表示するかを選択できます。ユーザーは有効なメニュー項目を選択できます。無効なメニュー項目は灰色で表示され、選択することはできません。選択可 チェックボックスの選択が解除されていると、メニュー命令は灰色で表示され、選択することができません。

明示的に設定しない限り、4D は自動でカスタムメニューに追加された項目を有効にします。たとえば、特定の条件下で **ENABLE MENU ITEM** コマンドを使用して有効化するために、初期状態を無効にすることができます (無効化には **DISABLE MENU ITEM** コマンドを使います)。

チェック

このオプションを使用して、メニュー項目にシステムチェックマークを関連付けることができます。その後チェックマークの表示をランゲージコマンド (**SET MENU ITEM MARK** や **Get menu item mark**) で制御できます。

通常チェックマークは連続したアクションをおこなうメニュー項目に付けられ、そのアクションを現在実行中であることを示すために使用されます。

フォントスタイル

メニュー命令にフォントスタイル (太字、下線、イタリック) を適用することができます。メニューエディターのオプションを使用して、または **SET MENU ITEM STYLE** ランゲージコマンドを使って、メニューのスタイルを太字・イタリック・下線でカスタマイズすることができます。

一般的なルールとして、フォントスタイルの適用は慎重におこなってください。煩雑なスタイルの使用はユーザーの注意をそらし、アプリケーションの見た目を悪くします。

メニュータイトルに制御文字を挿入してスタイルを管理することもできます ([制御文字の使用](#) 参照)。

項目アイコン

メニュー項目にアイコンを関連付けることができます。設定されたアイコンはメニューの左に表示されます:



メニューエディターでアイコンを設定するには、"項目アイコン" エリアをクリックし、開くを選択してディスクからピクチャーを開きます。プロジェクトの Resources フォルダーに格納されていないピクチャーファイルを選択した場合、そのファイルは自動的に Resources フォルダーにコピーされます。項目アイコンを設定すると、プレビューエリアに表示されます：



項目からアイコンを取り除くには、"項目アイコン" エリアのメニューから アイコンなし を選択します。

4Dランゲージを使って項目アイコンを設定するには、`SET MENU ITEM ICON` コマンドを使います。

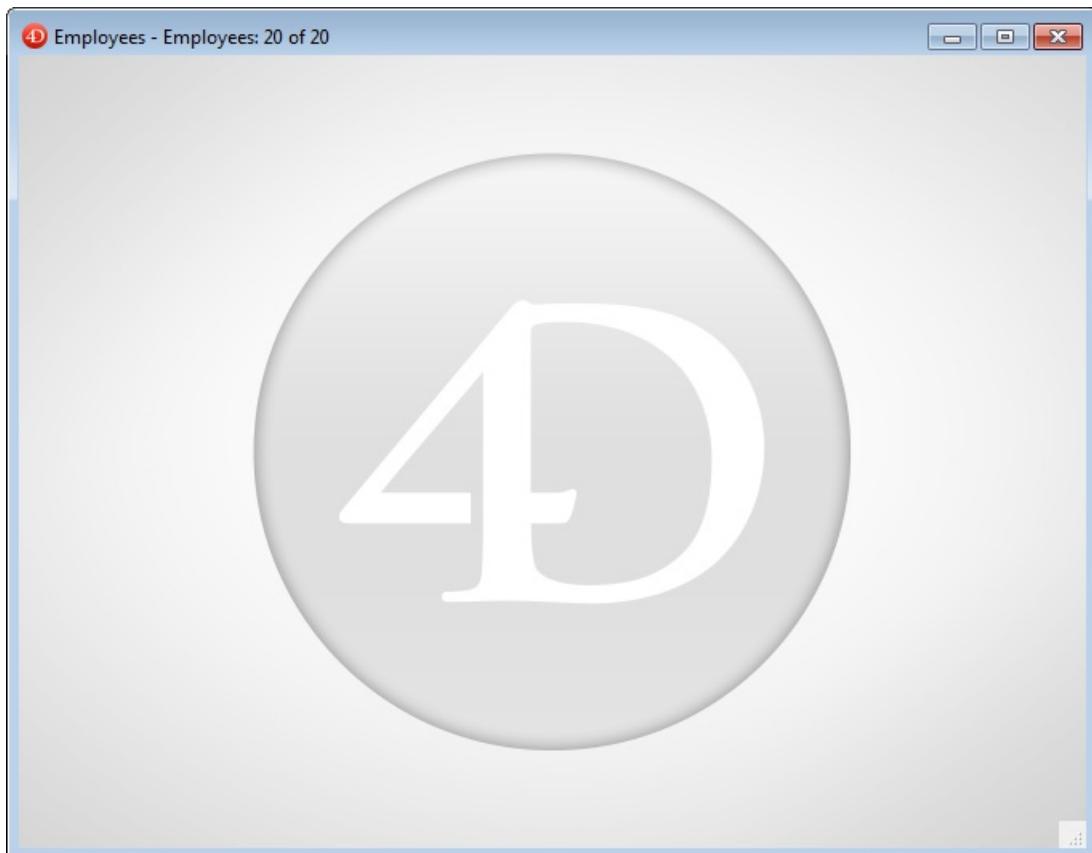
メニューバーの管理

メニューバーはカスタムアプリケーションにおいて主要なインターフェースを提供します。各カスタムアプリケーションにおいて、最低1つのメニューを添付したメニューバーを1つ作成しなければなりません。デフォルトで、メニューバー#1 がアプリケーションモードで表示されます。 SET MENU BAR コマンドを使用して、メニューバーを変更することができます。

各メニューバーにはカスタムスプラッシュスクリーンを関連付けることができます。またメニューバーとスプラッシュスクリーンはプレビューすることができます。

スプラッシュスクリーン

各メニューバーにカスタムスラッシュスクリーンを関連付けることにより、アピアランスを拡張できます。スプラッシュスクリーンを含むウィンドウは、メニューバーが表示されるとき、その下に表示されます。ロゴなどのピクチャーを表示できます。デフォルトで、4D はスプラッシュスクリーンに 4D ロゴを表示します：



任意の画像編集アプリケーションで作成したピクチャーをスプラッシュスクリーンで使用できます。クリップボードにコピーした画像、あるいはハードディスク上の画像を使用できます。4D がサポートする標準のピクチャータイプの画像を使用できます。

スプラッシュスクリーンピクチャーはメニューエディターでのみ設定できます：まず、カスタムスラッシュスクリーンを割り当てたいメニューバーを選択します。ウィンドウ右側に"背景画像"エリアが表示されます。ディスクに保存されたピクチャーを直接開くには、開く ボタンをクリックするか、"背景画像" エリアをクリックします。ポップアップメニューが表示されます：

- クリップボードのピクチャーをペーストするには ペースト を選択します。
- ディスクファイルとして保存された画像を開くには 開く を選択します。開くを選択すると、標準のファイルを開くダイアログボックスが表示されます。使用するピクチャーを選択します。設定が完了すると、選択した画像がプレビューとして表示されます。これにより、メニューバーとの関連付けが確認できます。



メニューバーをテストすると、設定の結果を見ることができます（後述参照）。アプリケーションモードでは、ピクチャーはスプラッシュスクリーンに "トランケート(中央合わせ)" で表示されます。

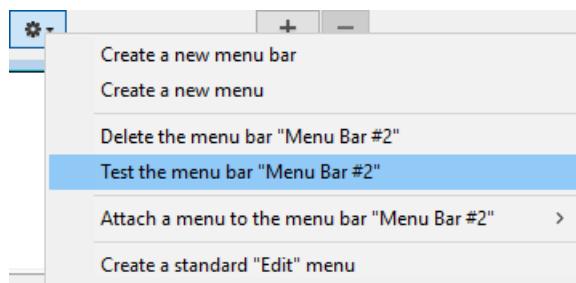
ストラクチャー設定では、インターフェース > 一般的 ウィンドウの表示 オプションを使用して、スプラッシュスクリーンの表示/非表示を設定できます。

カスタムピクチャーを削除してデフォルトに戻すには、クリア ボタンをクリックするか、エアーポップアップメニューから クリア を選択します。

メニューバーのプレビュー

メニューバー エディターからカスタムメニューとスプラッシュスクリーンをプレビューできます。ツールボックス ウィンドウを閉じる必要はありません。

これをおこなうにはメニューバーを選択して、コンテキストメニューまたはエディターのオプションメニューから、メニューバー "メニューバー #X" をテストを選択します。



すると、メニューバーとスプラッシュスクリーンのプレビューが表示されます。メニューや階層メニューを表示させることができます。ただし、プレビュー状態のメニューを選択してもコマンドは実行されません。メニュー やツールバー の動作を確認するには、実行 メニューから アプリケーションモード を選択します。

Windows での SDIモード

Windowsにおいて、組みこみ 4Dアプリケーションを SDI（シングルドキュメントインターフェース）アプリケーションとして設定することができます。SDIアプリケーションでは、それぞれのウィンドウが互いに独立し、それぞれが独自のメニューbaruを持つことができます。SDIアプリケーションは MDI（マルチドキュメントインターフェース）に対する概念で、MDI ではすべてのウィンドウが一つのメインウィンドウの中に含まれ、それに依存した作りになっています。

SDI/MDI という概念は macOS には存在しません。この機能は Windows用アプリケーション専用のもので、関連オプションは macOS においてはすべて無視されます。

SDIモード利用条件

SDIモードは以下の実行環境に限り利用可能です:

- Windows
- 組み込みスタンドアロン4Dアプリケーション、またはクライアント4Dアプリケーション

SDIモードの有効化

アプリケーションにおいて SDIモードを有効化し使用する手順は次の通りです:

1. ストラクチャー設定ダイアログボックスの "インターフェース" ページ内にある WindowsでSDIモードを使用するオプションをチェックします。
2. 組み込みアプリケーションをビルドします (スタンドアロンまたはクライアントアプリケーション)。

その後、サポートされているコンテキスト (上記参照) において実行されると、組み込みアプリケーションは自動的に SDIモードで実行されます。

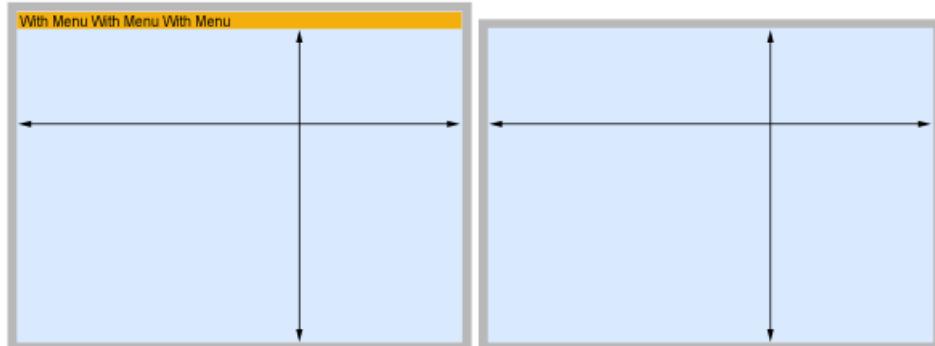
SDIモードでのアプリケーションの管理

4Dアプリケーションを SDIモードで実行するために、特別な実装は必要ありません。既存のメニューbaruは自動的に SDIウィンドウへと移されます。しかしながら、以下に挙げられている特定の原則に注意する必要があります。

ウィンドウ内のメニュー

SDIモードでは、同プロセス中に開かれたすべてのドキュメントタイプウィンドウ (たとえばフローティングパレットはこれに含まれません) には自動的にプロセスメニューbaruが表示されます。ただし、プロセスメニューbaruが非表示の状態でも、メニュー項目のショートカットは有効です。

メニューは、コンテンツのサイズを変更することなくウィンドウの上部に追加されます:



このため、ウィンドウは MDIモードあるいは SDIモードのどちらにおいてもオブジェクトの位置を再計算することなく使用することができます。

スプラッシュスクリーンについての注意:

- ストラクチャー設定において スプラッシュスクリーンインターフェースオプションが選択されていた場合、スプラッシュウィンドウは、MDIウィンドウであれば表示されていたメニューをすべて格納します。MDIモード同様、スプラッシュスクリーンを閉じるとアプリケーションを終了することになるという点に注意してください。

- スプラッシュスクリーンオプションが選択されていなかった場合、メニューは開かれているウィンドウにおいて、プログラマーの選択に応じて表示されます。

自動終了

MDIモードで実行時、ユーザーによってアプリケーションウィンドウ (MDIウィンドウ) が閉じられると、4Dアプリケーションが終了します。しかしながら、SDIモードで実行時、4Dアプリケーションにはアプリケーションウィンドウがなく、また開いているウィンドウをすべて閉じたとしても、必ずしもユーザーがアプリケーションを終了したいと思っているとは限りません（たとえばフェイスレスプロセスが熟考中かもしれません）が、場合によっては終了したいという場合もあります。

こういった場合を管理するため、SDIモードで実行されている4Dアプリケーションには、以下の条件が満たされた場合に自動的に（`QUIT 4D` コマンドを呼び出して）終了する機構が含まれています：

- ユーザーがこれ以上アプリケーションとやりとりすることができない
- 生きているユーザープロセスがない
- 4Dプロセスあるいはワーカープロセスはイベント待機中である
- Webサーバーが開始されていない

`quit`（終了）標準アクションが割り当てられているメニューが呼び出された場合、そのメニューがどこから呼ばれたものであろうと、アプリケーションは終了し、すべてのウィンドウが閉じられます。

ランゲージ

4Dによって透過的に管理されるとはいっても、SDIモードではアプリケーションインターフェースの管理に関してこれまでと若干の差異が存在します。4Dランゲージにおける特異性は以下の表にある通りです。

コマンド/機能	WindowsでのSDIモードの特徴
<code>Open form window</code>	SDIモードにおけるフローティングウィンドウのサポート（ <code>Controller form window</code> ）およびメニューバーの削除（ <code>Form has no menu bar</code> ）のオプション
<code>Menu bar height</code>	メニューバーが2行以上に折り返されている場合でも單一行のメニューバーのピクセル単位での高さを返します。フォームウィンドウをともなわないプロセスからコマンドが呼ばれている場合には0を返します。
<code>SHOW MENU BAR / HIDE MENU BAR</code>	カレントの（コードが実行されている場所の）フォームウィンドウにのみ適用されます
<code>MAXIMIZE WINDOW</code>	ウィンドウはスクリーンサイズいっぱいまで最大化されます
<code>CONVERT COORDINATES</code>	<code>XY Screen</code> はメインスクリーンが(0,0)に位置するグローバルな座標系です。座標系の左側、あるいは上側にあるスクリーンについては、負の値の座標を持つことができ、右側、あるいは下側にあるスクリーンについては <code>Screen height</code> や <code>Screen width</code> から返される値より大き値を持つことができます。
<code>GET MOUSE</code>	グローバル座標はスクリーンからの相対位置になります
<code>GET WINDOW RECT</code>	windowパラメーターに-1を渡した場合、コマンドは0;0;0;0を返します
<code>On Drop database method</code>	サポートされていません

ユーザー設定

4Dでは、プロジェクトの設定について 2つの運用モードが提供されています:

- 標準 モード :すべての [設定](#) は、[プロジェクトレベルの settings.4DSettings](#) ファイルに保存され、すべてのケースに適用されます。これはデフォルトのモードで、開発段階 (すべてのアプリケーション) に適しています。
- ユーザー設定 モード :カスタム設定の一部は、[Settingsフォルダー](#) (すべてのデータファイル用)、または [Dataフォルダー](#) (特定のデータファイル用) に置かれた [settings.4DSettings](#) ファイルに保存され、ストラクチャー設定の代わりに使用されます。このモードは、デスクトップアプリケーションの運用段階に適しています。このモードは、設定の [セキュリティページ](#) にあるオプションを使用して有効にします。

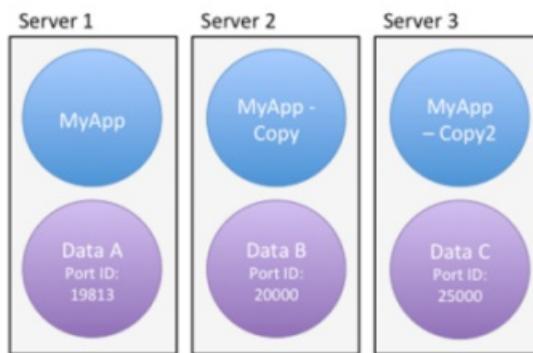
ユーザー設定を定義することで、4D アプリケーションを更新しても、カスタム設定を保持できるようになります。あるいは、異なる場所に展開する同じアプリケーションに対し、異なる設定を適用することが可能になります。また、設定ファイルの内容は XML で記述されるため、プログラムによる設定の管理もできるようになります。

4D は 2種類のユーザー設定を生成し使用することができます:

- ユーザー設定 (標準): これらのユーザー設定は、アプリケーションでどのデータファイルを開いたかにかかわらず、ストラクチャー設定の代わりに使用されます。
- データファイル用のユーザー設定: これらのユーザー設定は、アプリケーションで使用される各データファイルに対してそれぞれ関連づけられており、たとえばサーバーキャッシュのポートID などを設定します。

このオプションを使用すると、それぞれが異なる設定を持つデータファイルを複数使用する同じデスクトップアプリケーションのコピーを複数配布したりアップデートしたりすることが容易になります。

とあるアプリケーションが複製され、それぞれで異なる設定 (ポートID) を使用します。このユーザー設定がデータファイルとリンクしていた場合、ポートIDを手動で変えることなくアプリケーションをアップデートすることができます:



ユーザー設定の有効化

ユーザー設定を有効にするには、設定 > セキュリティ > 外部ファイルのユーザー設定を有効にする オプションを選択します:

Options

Filtering of commands and project methods in the formula editor and 4D Write Pro documents:

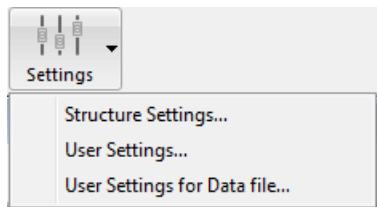
Enabled for all
 Disable for the Designer and the Administrator
 Disabled for all

Enable User Settings

このオプションをチェックすると、設定が 3つのダイアログに分けられます:

- ストラクチャー設定
- ユーザー設定
- データファイル用のユーザー設定

これらのダイアログボックスは、デザイン > 設定 メニュー、あるいはツールバーの 設定 ボタンからアクセスできます:

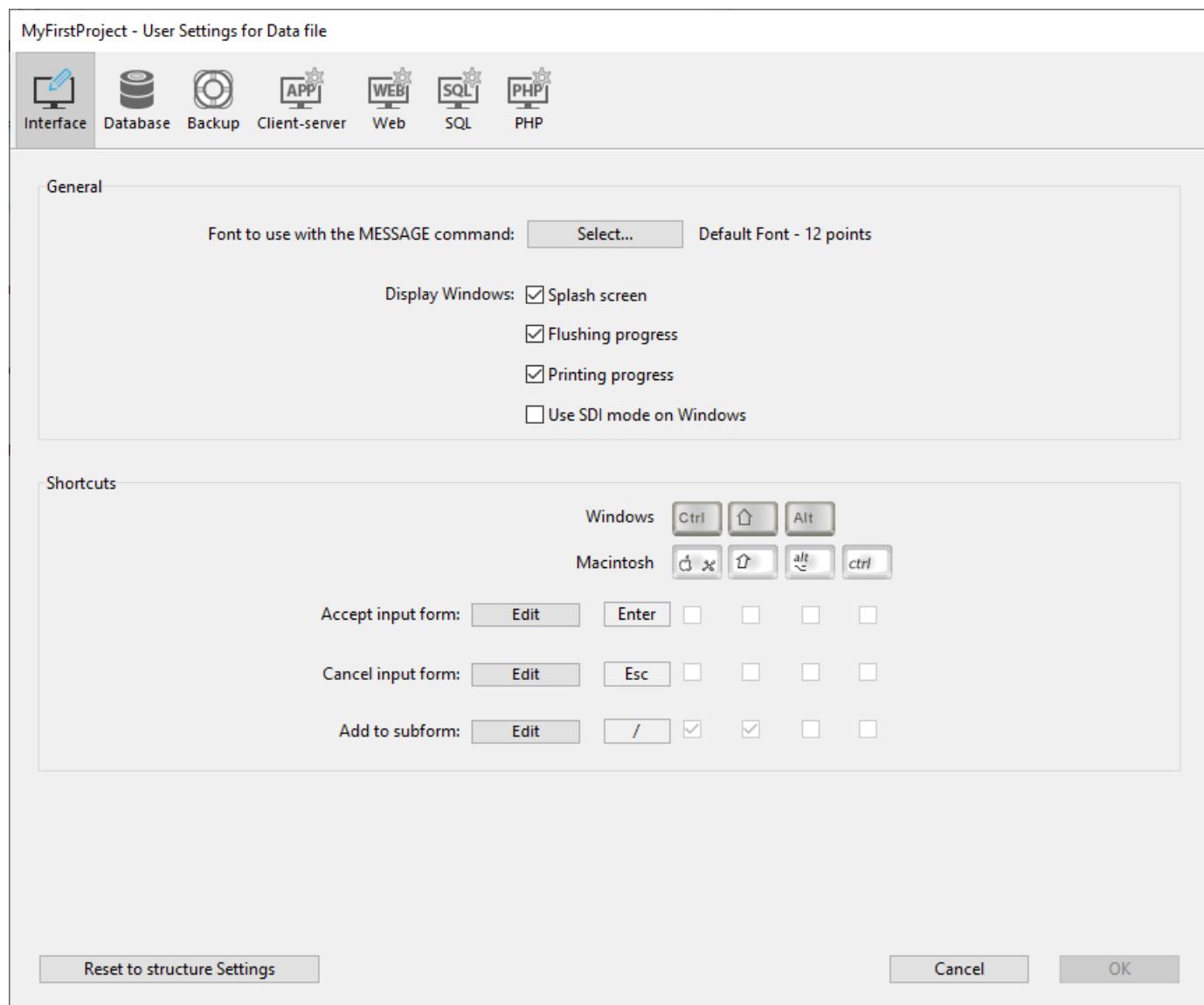


これらのダイアログボックスは、[OPEN SETTINGS WINDOW](#) コマンドに適切な *settingsType* セレクターを渡して使用することでもアクセスできます。

ストラクチャー設定ダイアログボックスは、標準の設定ダイアログと同じで、そのすべてのプロパティにアクセスできます（これらの設定はユーザー設定によってオーバーライドできます）。

ユーザー設定とデータファイル用のユーザー設定

ユーザー設定とデータファイル用のユーザー設定 ダイアログボックスには、すべてのデータファイルまたは 1つのデータファイルに対して定義できる関連プロパティが含まれています：



ユーザー設定およびデータファイル用のユーザー設定 ダイアログボックスに含まれる設定ページのリストと、標準設定との主な違いを以下の表にまとめます：

ストラクチャー設定のページ	ユーザー設定のページ	データファイル用のユーザー設定のページ
一般ページ	N/a	N/a
インターフェースページ	標準設定と同じ	標準設定と同じ
コンパイラーページ	N/a	N/a
データベース/データストレージページ	N/a	N/a
データベース/メモリページ	標準設定と同じ	標準設定と同じ
バックアップ/スケジューラーページ	N/a	標準設定と同じ
バックアップ/設定ページ	N/a	標準設定と同じ
バックアップ/バックアップ & 復旧ページ	N/a	標準設定と同じ
クライアント-サーバー/ネットワークオプションページ	標準設定と同じ	標準設定と同じ
クライアント-サーバー/IP設定ページ	標準設定と同じ	標準設定と同じ
Web/設定ページ	標準設定と同じ	標準設定と同じ
Web/オプション (I) ページ	標準設定と同じ	標準設定と同じ
Web/オプション (II) ページ	標準設定と同じ	標準設定と同じ
Web/ログ (タイプ) ページ	標準設定と同じ	標準設定と同じ
Web/ログ (バックアップ) ページ	標準設定と同じ	標準設定と同じ
Web/Webサービスページ	メソッドプリフィックスオプションは使用不可	メソッドプリフィックスオプションは使用不可
SQL ページ	標準設定と同じ	標準設定と同じ
PHP ページ	標準設定と同じ	標準設定と同じ
セキュリティページ	N/a	N/a
互換性ページ	N/a	N/a

このダイアログボックスでの設定を編集した場合、それらの変更は対応する `settings.4DSettings` ファイルに自動的に保存されます（後述参照）。

SET DATABASE PARAMETER とユーザー設定

ユーザー設定の一部は `SET DATABASE PARAMETER` コマンドでも利用可能です。ユーザー設定は、2セッション間で設定を保持 プロパティが Yes になっているパラメーターです。

ユーザー設定 機能が有効化されている場合、`SET DATABASE PARAMETER` コマンドで編集されたユーザー設定はデータファイル用のユーザー設定に自動的に保存されます。

`Table sequence number` は例外です。この設定値は常にデータファイル自身に保存されます。

settings.4DSettings ファイル

データベース設定において [外部ファイルのユーザー設定を有効にするオプションをチェックした場合](#)、ユーザー設定ファイルは自動的に作成されます。これらのファイルの場所は、ユーザー設定の種類に応じて決まります。

ユーザー設定 (標準)

標準のユーザー設定ファイルは自動的に作成され、以下の場所にある Settings フォルダー内に保存されます：

`ProjectFolder/Settings/settings.4DSettings`

`ProjectFolder` は、プロジェクトストラクチャーファイルが格納されているフォルダーの名前です。

組み込みアプリケーションでは、ユーザー設定ファイルは以下の場所に配置されます：

- ・ シングルユーザー版の場合: ProjectFolder/Database/Settings/settings.4DSettings
- ・ クライアント-サーバー版の場合: ProjectFolder/Server Database/Settings/settings.4DSettings

データファイル用のユーザー設定

データファイルにリンクされているユーザー設定ファイルは自動的に作成され、以下の場所にある Settings フォルダー内に保存されます:

Data/Settings/settings.4DSettings

Data は、アプリケーションのカレントデータファイルが格納されているフォルダーの名前です。

データファイルがプロジェクトストラクチャーファイルと同階層に位置している場合、ストラクチャー用とデータ用のユーザー設定ファイルは同じ場所の同じファイルを共有します。データファイル用のユーザー設定... メニューは表示されません。

設定ファイルは XML ファイルであり、4D の XML コマンドや、XML エディターを使用して読み込んだり変更したりすることができます。つまり、特にコンパイルされて 4D Volume Desktop と組み込まれたアプリケーションにおいて、設定内容をプログラムで管理することが可能です。このファイルをプログラムで編集した場合、変更内容が反映されるのはデータベース再起動後です。

設定の優先順位

設定は 3 つの階層に保存することができます。ある階層で定義されたそれぞれの設定は、前のレベルで定義された設定を上書きします（あれば）：

優先度	名称	場所	コメント
3 (低)	ストラクチャー設定 (あるいは、"ユーザー設定" 機能無効時の設定)	Sources フォルダー内の settings.4DSettings ファイル (プロジェクトモードデータベースの場合)、またはストラクチャーファイルと同階層にある Settings フォルダー内 (バイナリーモードデータベースの場合)	ユーザー設定が有効化されていない場合の固有の保存位置。アプリケーションの複製すべてに適用。
2	ユーザー設定 (全データファイル)	プロジェクトフォルダーと同階層にある Settings フォルダー内の settings.4DSettings ファイル	ストラクチャー設定を上書きします。アプリケーションパッケージ内に保存されます。
1 (高)	ユーザー設定 (カレントデータファイル)	データファイルと同階層にある Settings フォルダー内の settings.4DSettings ファイル	ストラクチャー設定とユーザー設定を上書きします。その設定とリンクされたデータファイルがアプリケーションによって使用されたときにのみ適用。

ユーザー設定ファイルには関連した設定の一部しか含まれない一方、ストラクチャーファイルには、コア設定を含めたカスタム設定がすべて格納されているという点に注意してください。

アプリケーションビルド

4D にはプロジェクトパッケージ (ファイナルビルド) を作成するためのアプリケーションビルダーが統合されています。このビルダーを使用すれば、コンパイルされた 4D アプリケーションの展開を簡易化することができます。OS ごとに異なる特定の処理を自動で処理し、クライアント/サーバーアプリケーションの展開が容易になります。

アプリケーションビルダーでは以下のことを行えます:

- インターブリターコードを含まないコンパイル済みストラクチャーのビルド
- ダブルクリックで起動可能なスタンドアロンアプリケーションのビルド (4D のデータベースエンジンである 4D Volume Desktop を組み込んだ 4D アプリケーション)
- XML形式のプロジェクトファイル定義を用いて、同じコンパイル済みストラクチャーから異なるアプリケーションのビルド
- クライアント/サーバーアプリケーションのビルド
- クライアントとサーバーの自動更新機能を備えたクライアント/サーバーアプリケーションのビルド
- ビルド設定の保存 (設定保存 ボタン)

コンパイル済みアプリケーションは、読み取り専用 である `.4dz files` ファイルに基づきます。コンパイル済みアプリケーションの場合、ソースファイルを変更するコマンドや関数 (`CREATE INDEX` や `CREATE TABLE` (SQL)) は、デフォルトでは使用できないことに留意が必要です。しかしながら、`PackProject` XML キー ([doc.4d.com 参照](#)) を使用することで、ローカルな変更をサポートするアプリケーションをビルドすることも可能です。

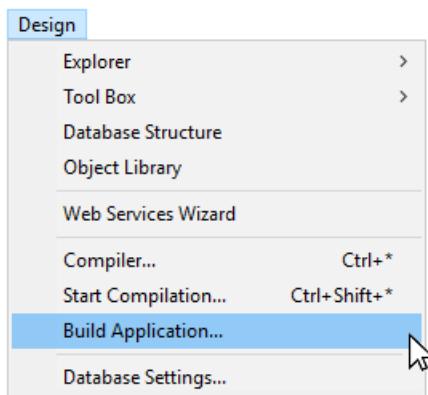
概要

プロジェクトパッケージをビルドするには次の方法があります:

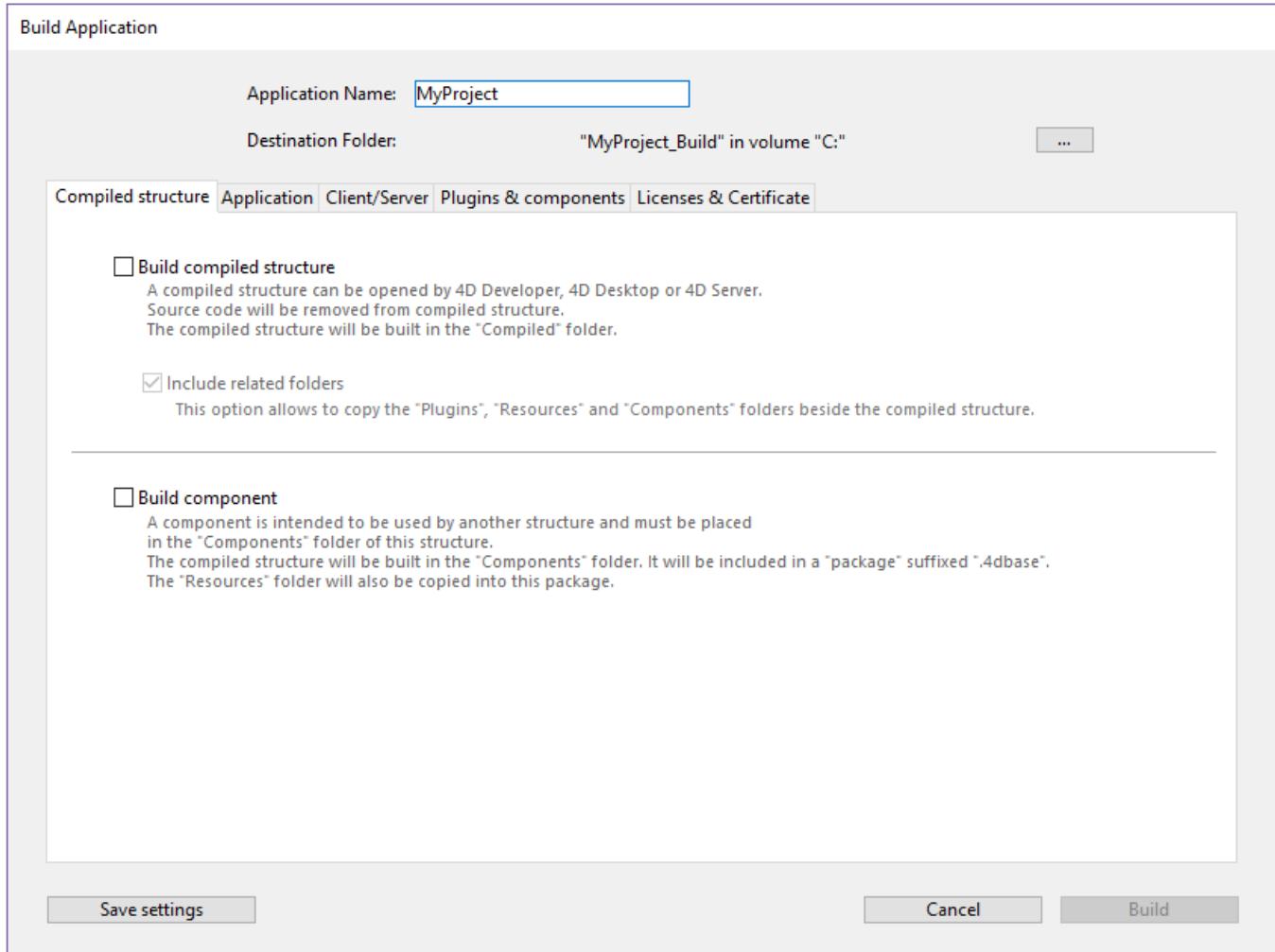
- `BUILD APPLICATION` コマンドを使う
- アプリケーションビルド ダイアログを使う

アプリケーションビルド

このウィンドウを開くには 4D のデザインメニューからアプリケーションビルド...を選択します。



アプリケーションビルドウィンドウには複数のページがあり、タブを使用してページを移動できます:



ビルドをおこなう前にプロジェクトはコンパイルされていなければなりません。まだコンパイルされていないプロジェクトでこのメニュー命令を選択する、あるいはコンパイル後にコードが変更されていると、プロジェクトを（再）コンパイルしなければならない旨の警告ダイアログが表示されます。

buildApp.4DSettings

また、Windows で保護されたファイルを使用できるように、インストール権限を昇格させる XMLキーもあります ([アプリケーションビルト設定ファイル](#) のマニュアルを参照ください)。

アプリケーションビルトダイアログが初めて表示されるときにはデフォルトパラメーターが使用されます。ビルト ボタンや 設定保存 ボタンをクリックすると、このプロジェクトファイルの内容が更新されます。同じデータベースについて内容の異なる複数の XML ファイルを定義し、[BUILD APPLICATION](#) コマンドでそれらを使い分けることができます。

また、XML キーを使用すれば、アプリケーションビルトダイアログには表示されない追加の設定をおこなうことができます。詳細は専用のドキュメント [アプリケーションビルト設定ファイル](#) を参照してください。

ログファイル

アプリケーションをビルドすると、4D はログファイル (*BuildApp.log.xml*) を生成して、プロジェクトの Logs フォルダーに保存します。ログファイルにはビルド毎に以下の情報が書き込まれます：

- ターゲットビルドの開始と終了
- 生成されたファイルの名称とフルパス
- ビルドの日付と時刻
- 発生したエラー
- 署名の問題 (例：署名されていないプラグイン)

アプリケーションを公証する場合などは、このファイルを確認することで、後の運用手順で時間を節約することができます。

`Get 4D file(Build application log file)` コマンドを使って、ログファイルの場所を取得します。

アプリケーション名と保存先フォルダー

Application Name:

Destination Folder: "MyProject_Build" in volume "C:"

アプリケーション名 には生成するアプリケーションの名前を入力します。

保存先フォルダー にはビルドされるアプリケーションの保存先を指定します。指定したフォルダーが存在しない場合は新たに作成します。

コンパイル済みストラクチャーページ

このページでは、標準のコンパイル済みストラクチャーファイルやコンパイル済みコンポーネントをビルドできます。

Build Application

Application Name:

Destination Folder: "MyProject_Build" in volume "C:"

Compiled structure Application Client/Server Plugins & components Licenses & Certificate

Build compiled structure
A compiled structure can be opened by 4D Developer, 4D Desktop or 4D Server.
Source code will be removed from compiled structure.
The compiled structure will be built in the "Compiled" folder.

Include related folders
This option allows to copy the "Plugins", "Resources" and "Components" folders beside the compiled structure.

Build component
A component is intended to be used by another structure and must be placed
in the "Components" folder of this structure.
The compiled structure will be built in the "Components" folder. It will be included in a "package" suffixed ".4dbase".
The "Resources" folder will also be copied into this package.

コンパイル済みストラクチャーをビルド

インタープリターコードを含まないアプリケーションをビルドします。

これにより、*Compiled Database/<project name>* フォルダーの中に .4dz ファイルが作成されます。たとえば、アプリケーション名を "MyProject" にした場合、4D は次のものを作成します:

<destination>/Compiled Database/MyProject/MyProject.4dz

.4dz ファイルは ZIP 圧縮されたプロジェクトフォルダーです（注：バイナリデータベースの場合に生成される .4DC ファイルと同義ではないことに注意が必要です）。.4dz ファイルを開けるのは 4D Server、4D Volume ライセンス（組み込みアプリケーション）、および 4D です。圧縮・最適化された .4dz ファイルによってプロジェクトパッケージの展開が容易になります。

.4dz ファイルを生成する際、4D はデフォルトで 標準的な zip 形式を使用します。このフォーマットの利点は、あらゆる解凍ツールで簡単に読み取ることができます。この標準形式を使用したくない場合は、値を `False` に設定した `UseStandardZipFormat` XML キーを `buildApp.4DSettings` ファイルに追加します（詳細については、[アプリケーションビルド設定ファイル](#) マニュアルを参照ください）。

関連するフォルダーを含む

このオプションを選択すると、プロジェクトに関連するフォルダーが、Build フォルダーの *Components* および *Resources* フォルダーにコピーされます。これらのフォルダーの詳細については [プロジェクトアーキテクチャーの説明](#) を参照ください。

コンポーネントをビルド

ストラクチャーからコンパイル済みコンポーネントをビルドします。

コンポーネントは特定の機能を実装した標準の 4D プロジェクトです。ビルドされたコンポーネントを他の 4D プロジェクト（ホストアプリケーションプロジェクト）にインストールすると、ホストプロジェクトはその機能を利用できるようになります。

アプリケーション名を *MyComponent* に指定した場合、4D は *Components* フォルダーを作成し、その中に *MyComponent.4dbase* フォルダーを生成します：

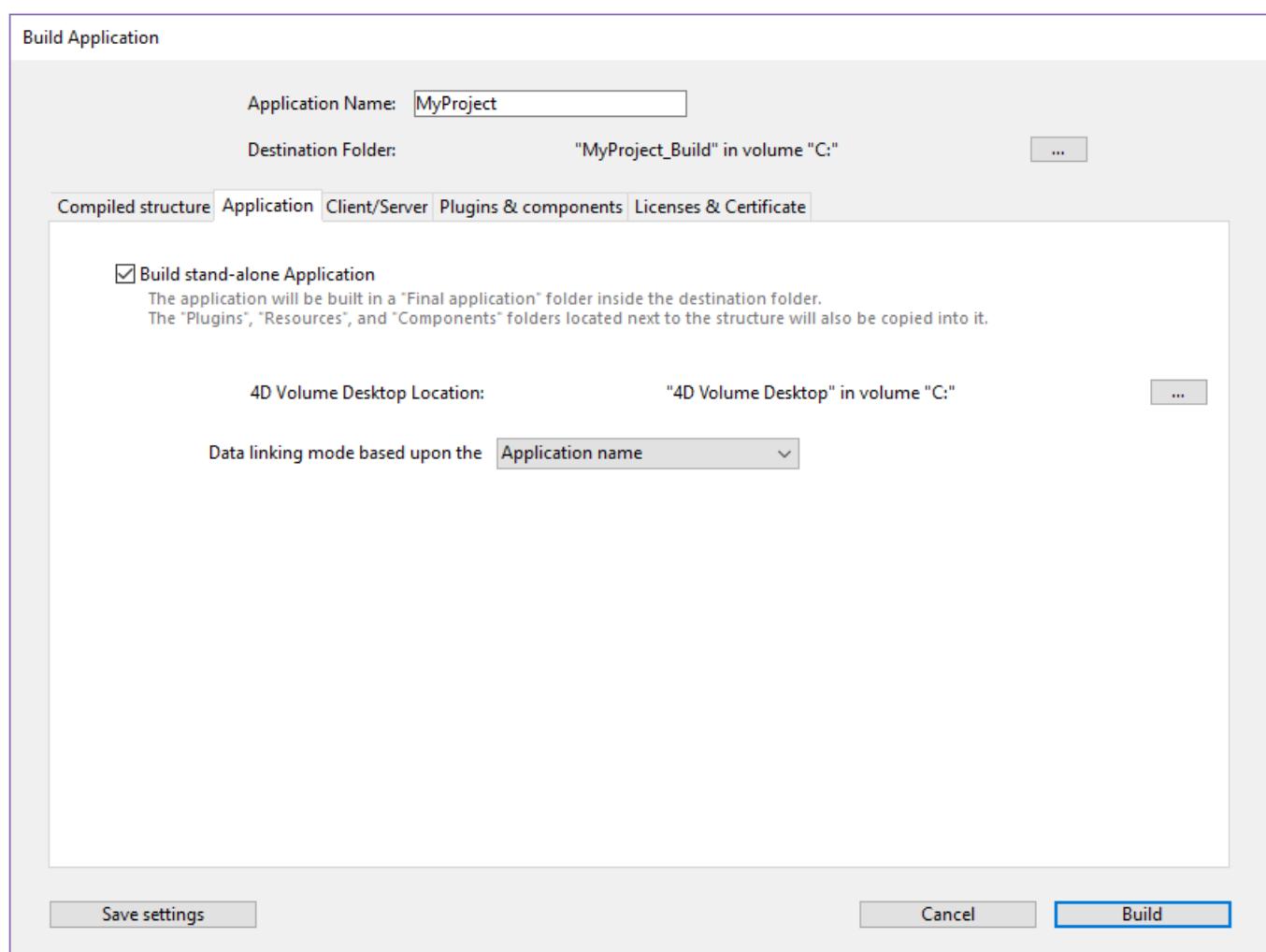
<destination>/Components/MyComponent.4dbase/MyComponent.4DZ.

MyComponent.4dbase フォルダーには次のファイルが含まれます：

- *MyComponent.4DZ* ファイル
- *Resources* フォルダー：関連リソースは自動的にこのフォルダーにコピーされます。コンポーネントは、他のコンポーネントやプラグインを使用できないため、その他の "Components" や "Plugins" フォルダーはコピーされません。

アプリケーションページ

このタブでは、スタンドアロンのシングルユーザー版アプリケーションをビルドします：



スタンドアロンアプリケーションをビルド

スタンドアロンアプリケーションをビルド オプションを選択して ビルド ボタンをクリックすると、スタンドアロンの（つまり、ダブルクリックで起動可能な）アプリケーションがアプリケーションプロジェクトをもとに作成されます。

ビルドには次のものが必要です:

- 4D Volume Desktop (4Dデータベースエンジン)
- 適切な [ライセンス](#)

Windowsにおいては、.exe 拡張子のついた実行ファイルが作成されます。macOSにおいては、ソフトウェアパッケージが作成されます。

この処理はコンパイル済みストラクチャーファイルと4D Volume Desktopを統合します。4D Volume Desktop が提供する機能はライセンスページで指定するライセンス情報に基づきます。この点についての詳細な情報は、4D の [オンラインストア](#) と、セールスドキュメンテーションを参照してください。

データファイルについては、デフォルトのデータファイルを定義することもできますし、ユーザー独自のデータファイルを作成・使用することもできます (詳細については [最終アプリケーションでのデータファイルの管理](#) を参照してください)。

いくつかのランゲージコマンドを特定の順番で使用することによって、シングルユーザー向け組み込みアプリケーションのアップデートを自動化することが可能です (サーバーまたはシングルユーザー向けアプリの自動アップデートを参照してください)。

4D Volume Desktopの場所

ダブルクリックで起動されるアプリケーションをビルドするには、まず 4D Volume Desktop が格納されているフォルダーの場所を指定しなければなりません:

- Windows では: 4D Volume Desktop.4DE や 4D Volume Desktop.RSR、その他動作に必要なファイルやフォルダーを含むフォルダーを選択します。これらは、選択されたフォルダー内で同じ階層に置かれている必要があります。
- macOS では: ソフトウェアパッケージとして 4D Volume Desktop が提供されているので、このパッケージを選択します。

4D Volume Desktop フォルダーを選択するには [...] ボタンをクリックします。フォルダーを選択するダイアログが表示されたら、4D Volume Desktop フォルダー (Windows) またはパッケージ (macOS) を選択します。

フォルダーが選択されるとその完全パス名が表示され、そこに 4D Volume Desktop が含まれていればビルドボタンが有効になります。

4D Volume Desktop のバージョン番号は、4D Developer のバージョン番号と合致する必要があります。たとえば、4D Developer の v18 を利用していれば、4D Volume Desktop v18 が必要です。

データリンクモードの基準

このオプションを使って、組み込みアプリケーションとローカルデータファイルとのリンクモードを選択します。二種類のリンクモードから選択可能です:

- アプリケーション名 (デフォルト) - このモードでは、4D アプリケーションはストラクチャーファイルに対応する、最後に開かれたデータファイルを開きます。このモードではアプリケーションパッケージをディスク上で自由に移動させることができます。アプリケーションを複製する場合を除いて、通常は組み込みアプリに対してこのモードが使用されるべきです。
- アプリケーションパス - このモードでは、組み込み 4D アプリケーションは自身に紐づいている *lastDataPath.xml* ファイルを解析して、起動アプリのフルパスに合致する "executablePath" 属性を持つデータパスマップのエントリーを探し、同エントリー内で "dataFilePath" 属性で定義されているデータファイルを開きます。ない場合は、最後に開かれたデータファイルを開きます (デフォルトモード)。

データリンクモードについての詳細は [最後に開かれたデータファイル](#) を参照してください。

生成されるファイル

ビルド ボタンをクリックすると、4D は 保存先フォルダー に Final Application フォルダーを作成し、その中に指定したアプリケーション名のサブフォルダーを作成します。

アプリケーション名に "MyProject" と指定した場合、MyProject サブフォルダー内には以下のファイルが置かれます:

- Windows
 - MyProject.exe - 実行可能ファイル、そして MyProject.rsr (アプリケーションリソースファイル)
 - 4D Extensions および Resources フォルダー、さまざまなライブラリ (DLL)、Native Components フォルダー、SASL Plugins フォルダーなど、アプリケーション実行に必要なファイル
 - Databaseフォルダー: Resources フォルダーと MyProject.4DZ ファイルが格納されています。これらはプロジェクトのコンパイル済みストラクチャーおよびプロジェクトの Resources フォルダーです。注: このフォルダには、定義されていれば Default Data フォルダーも含まれています ([最終アプリケーションでのデータファイルの管理](#) を参照してください)。
 - (オプション) データベースに含まれるコンポーネントやプラグインが配置された Components フォルダーおよび Plugins フォルダー。この点に関する詳細は [プラグイン&コンポーネントページ](#) を参照してください。
 - Licenses フォルダー - アプリケーションに統合されたライセンス番号の XML ファイルが含まれます。この点に関する詳細は [ライセンス&証明](#)

[書ページ](#) を参照してください。

- 4D Volume Desktop フォルダーに追加されたその他の項目 (あれば) ([4D Volume Desktop フォルダーのカスタマイズ](#) 参照)

実行ファイルの動作には、これらすべての項目が同じフォルダー内に必要です。

- macOS

- MyProject.app という名称のソフトウェアパッケージに、プラグインやコンポーネント、ライセンスなど必要な項目がすべて格納されます。プラグインやコンポーネントの統合に関する詳細は [プラグイン & コンポーネントページ](#) を参照してください。ライセンスの統合に関しては [ライセンス & 証明書ページ](#) を参照してください。注: macOSでは、4D ランゲージの [Application file](#) コマンドが返すのは、ソフトウェアパッケージ内の "Contents:macOS" フォルダー内にコピーされる ApplicationName ファイルのパス名です (ソフトウェアパッケージ内の "Contents:Resources" フォルダー内の .comp ファイルのパスではありません)。

4D Volume Desktop フォルダーのカスタマイズ

ダブルクリックで起動可能なアプリケーションをビルドする際、4D は 4D Volume Desktop フォルダーの内容を *Final Application* 内のアプリケーション名サブフォルダーにコピーします。必要に応じて、このコピー元である 4D Volume Desktop フォルダーの内容をカスタマイズすることができます。たとえば:

- 特定の言語バージョンに対応する 4D Volume Desktop をインストールする
- カスタムプラグインを *Plugins* フォルダーに置く
- *Resources* フォルダーの内容をカスタマイズする

macOS では、4D Volume Desktop はソフトウェアパッケージ形式で提供されています。内容を変更するにはパッケージを開きます (アイコンを Control+click)。

Webファイルの場所

ダブルクリックで起動可能なアプリケーションを Webサーバーとして使用する場合、Web フォルダーやファイルは特定の場所にインストールする必要があります :

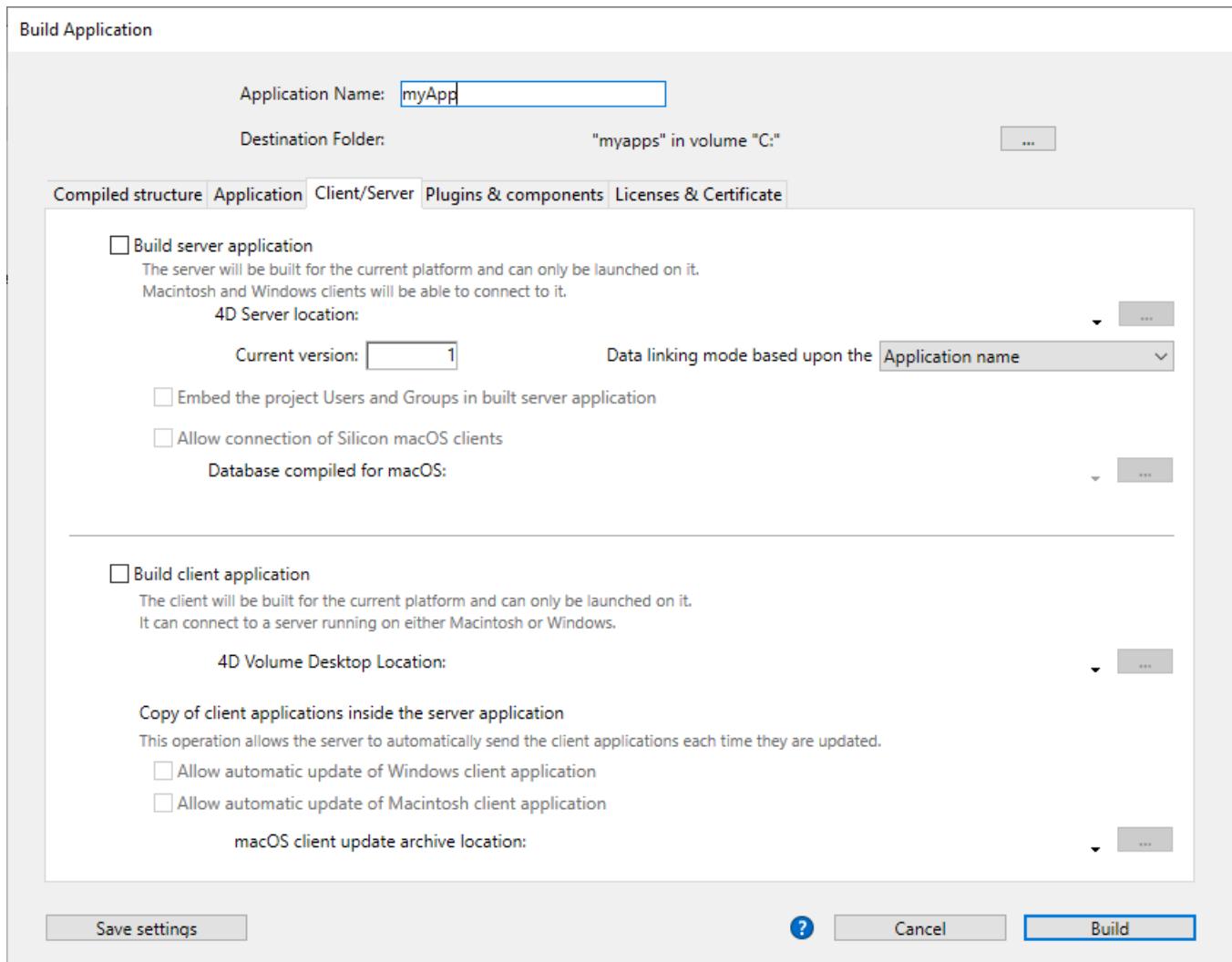
- *cert.pem* と *key.pem* ファイル (オプション): これらのファイルはTLS接続とデータ暗号化コマンドに使用されます。
- デフォルト Web ルートフォルダー

インストール場所:

- Windows: *Final Application*¥MyProject¥Database サブフォルダー内
- macOS: *MyProject.app* ソフトウェアパッケージと同階層

クライアント/サーバーページ

このページでは、クライアントの自動更新もサポートできるクロスプラットフォームなクライアント/サーバーアプリケーションをビルドするための設定をおこないます。



クライアント/サーバーアプリケーションとは

クライアント/サーバーアプリケーションは、以下の3つの項目の組み合わせから成ります：

- コンパイルされた 4Dプロジェクト
- 4D Server アプリケーション
- 4D Volume Desktop アプリケーション (macOS / Windows)

ビルドを行うと、クライアント/サーバーアプリケーションは 2つのカスタマイズされたパート（サーバーと、各クライアントマシンにインストールするクライアント）で構成されます。

Intel/AMD と Apple Silicon マシンが混在している環境でクライアントアプリケーションを実行する場合、そのクライアントサーバーアプリケーションは macOS 上で、[全てのプロセッサ](#) 向けにコンパイルすることが推奨されます。

ビルドされたクライアント/サーバーアプリケーションは起動や接続処理が簡易です：

- サーバーを起動するには、サーバーアプリケーションをダブルクリックします。プロジェクトファイルを選択する必要はありません。
- クライアントを起動するにも、同様にクライアントアプリケーションをダブルクリックします。すると、サーバーアプリケーションへの接続が直接おこなわれるため、接続ダイアログでサーバーを選択する必要はありません。クライアントは接続対象のサーバーを名称（サーバーが同じサブネットワーク上にある場合）、あるいはIPアドレスによって認識します。IPアドレスの指定は buildapp.4DSettings ファイル内の IPAddress XMLキーを使用して設定されます。接続が失敗した場合のために、代替機構を実装することができます。これについては [クライアント接続の管理](#) の章で説明されています。また、Option (macOS) や Alt (Windows) キーを押しながらクライアントアプリケーション起動すると、標準の接続ダイアログを強制的に表示させることもできます。サーバーアプリケーションには、対応するクライアントアプリケーションのみが接続できます。標準の 4Dアプリケーションを使用してサーバーアプリケーションに接続を試みると、接続は拒否されエラーが返されます。
- クライアント側を [ネットワーク越しに自動更新](#) するようにクライアント/サーバーアプリケーションを設定することも可能です。クライアントアプリケーションは最初のバージョンのみビルドして配布する必要があります。以降のアップデートは、自動アップデート機構を利用することで管理します。
- また、ラengeージコマンド ([SET UPDATE FOLDER](#)、および [RESTART 4D](#)) を使用して、サーバーアプリケーションの更新を自動化することも可能です

サーバーアプリケーションをビルド

アプリケーションのサーバー部分をビルドするにはこのオプションを選択します。ビルドに使用する 4D Server アプリケーションの場所を選択する必要があります。この 4D Server はビルドをおこなうプラットフォームに対応していなければなりません（たとえば、Windows 用のサーバーアプリケーションをビルドするには Windows 上でビルドを実行する必要があり、Windows 版の 4D Server アプリケーションを指定する必要があります）。

4D Server の場所

4D Server フォルダーを選択するには [...] ボタンをクリックします。macOS では 4D Server パッケージを選択します。

現在のバージョン

生成されるアプリケーションのバージョン番号を指定します。このバージョン番号をもとに、クライアントアプリケーションからの接続を受け入れたり拒否したりできます。クライアントとサーバーアプリケーションで互換性のある番号の範囲は [XMLキー](#) で設定します。

ビルドしたサーバーアプリケーションにプロジェクトのユーザーとグループを埋め込む

注記：ここでは、以下の用語を使用します：

名称	定義
プロジェクトのディレクトリファイル	プロジェクトの Settings フォルダー に置かれた <code>directory.json</code> ファイル
アプリケーションのディレクトリファイル	ビルドされた 4D Server の Settings フォルダー に置かれた <code>directory.json</code> ファイル
データのディレクトリファイル	Data > Settings フォルダー に置かれた <code>directory.json</code> ファイル

このオプションをチェックすると、ビルド時にプロジェクトのディレクトリファイルがアプリケーションのディレクトリファイルとしてコピーされます。

ビルドした 4D Server アプリケーションを実行すると：

- データのディレクトリファイルがサーバーにある場合は、それがロードされます。
- データのディレクトリファイルがサーバーにない場合は、アプリケーションのディレクトリファイルがロードされます。

アプリケーションのディレクトリファイルは読み取り専用です。ユーザー・グループ・パーミッションに対してサーバー実行中におこなわれた変更は、データのディレクトリファイルに保存されます。データのディレクトリファイルが存在しない場合は、自動作成されます。アプリケーションのディレクトリファイルが埋め込まれている場合は、データのディレクトリファイルとして複製されます。

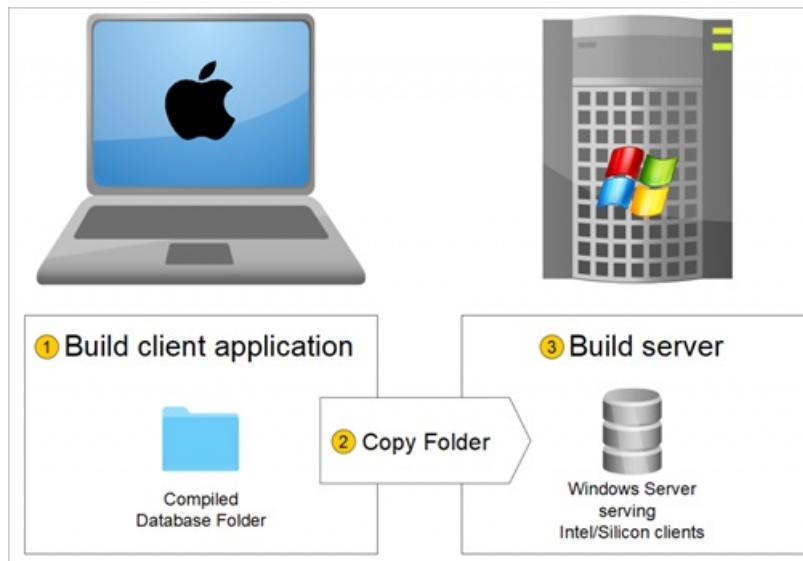
プロジェクトのディレクトリファイルを埋め込むことで、基本的なセキュリティのユーザーとグループ構成でクライアント/サーバーアプリケーションを運用することができます。その後の編集は、データのディレクトリファイルに追加されます。

Silicon macOS クライアントからの接続を許可

Windows でサーバーアプリケーションをビルドする際、Apple Silicon クライアントが接続できるようにするには、このオプションをチェックします。その後、Apple Silicon/Intel用にコンパイルされたストラクチャーへのパスを指定します。

Windows でビルドされたサーバーアプリケーションに Apple Silicon クライアントが接続できるようにするには、まず macOS 上で Apple Silicon と Intel 向けにコンパイルされたプロジェクトを用いてクライアントアプリケーションをビルドする必要があります。これにより、[コンパイル済みストラクチャーをビルド](#) オプションで作成したものと同じコンパイル済みストラクチャーが自動的に作成されます（関連フォルダーは含まれません）。

その後、そのストラクチャーを Windows マシンにコピーし、それを使ってサーバーアプリケーションをビルドすることができます：



コンパイル済みストラクチャーの場所

Windows用サーバーアプリケーションのビルドに使用される Apple Silicon/Intel クライアントアプリケーションのコンパイル済みストラクチャーへのパス ([Silicon macOS クライアントからの接続を許可 参照](#))。

データリンクモードの基準

このオプションを使って、組み込みアプリケーションとローカルデータファイルとのリンクモードを選択します。二種類のリンクモードから選択可能です：

- アプリケーション名 (デフォルト) - このモードでは、4D アプリケーションはストラクチャーファイルに対応する、最後に開かれたデータファイルを開きます。このモードではアプリケーションパッケージをディスク上で自由に移動させることができます。アプリケーションを複製する場合を除いて、通常は組み込みアプリに対してこのモードが使用されるべきです。
- アプリケーションパス - このモードでは、組み込み 4D アプリケーションは自身に紐づいている *lastDataPath.xml* ファイルを解析して、起動アプリのフルパスに合致する "executablePath" 属性を持つデータパスマップのエントリーを探し、同エントリー内で "dataFilePath" 属性で定義されているデータファイルを開きます。ない場合は、最後に開かれたデータファイルを開きます (デフォルトモード)。

データリンクモードについての詳細は [最後に開かれたデータファイル](#) を参照してください。

クライアントアプリケーションをビルド

アプリケーションのクライアント部分をビルドするにはこのオプションを選択します。

このオプションをチェックすると同時に：

- **サーバーアプリケーションをビルド** オプションを選択：現在のプラットフォーム用のサーバーと対応するクライアントをビルドし、(任意で) 自動アップデートアーカイブファイルも含むことができます。
- **サーバーアプリケーションをビルド** オプションを選択しない：通常は、サーバーのビルド時に選択する "別プラットフォーム" 用のアップデートアーカイブをビルドするときにこの設定を使います。

4D Volume Desktopの場所

クライアントアプリケーションのビルドに使用する 4D Volume Desktop アプリケーションの場所を指定します。

4D Volume Desktop のバージョン番号は、4D Developer のバージョン番号と合致する必要があります。4D Volume Desktop のバージョン番号は、4D Developer のバージョン番号と合致する必要があります。

この 4D Volume Desktop はビルドをおこなうプラットフォームに対応していないかもしれません。異なるプラットフォーム用のクライアントアプリケーションをビルドするには、そのプラットフォームで 4D アプリケーションを実行し、追加のビルド処理をしなければなりません。

クライアントアプリから特定のアドレスを使用して (サブネットワーク上にサーバー名が公開されていない) サーバーに接続したい場合、`buildapp.4DSettings` ファイル内の `IPAddress` XML キーを使用する必要があります。詳細は [BUILD APPLICATION](#) コマンドの説明を参照してください。接続失敗時の特定の機構を実装することもできます。詳細は [クライアント接続の管理](#) で説明されています。

サーバーアプリケーション内部のクライアントアプリケーションのコピー

このエリアのオプションは、クライアント/サーバーアプリケーションの新しいバージョンがビルドされた際の、ネットワーク越しにクライアントを自動更新するメカニズムを設定します。これらのオプションは、クライアントアプリケーションをビルト オプションがチェックされている場合にのみ有効です。

- Windows クライアントアプリケーションの自動更新を有効にする - このオプションをチェックすると、アップデートの際に Windows プラットフォーム上のクライアントアプリケーションに送信される .4darchive ファイルを作成します。
- macOS クライアントアプリケーションの自動更新を有効にする - このオプションをチェックすると、アップデートの際に macOS プラットフォーム上のクライアントアプリケーションに送信される .4darchive ファイルを作成します。

.4darchive は以下の場所にコピーされます:

```
<ApplicationName>_Build/Client Server executable/Upgrade4DClient/
```

別プラットフォームのクライアントアーカイブの選択

別プラットフォーム上で動作するクライアントアプリケーション用に、自動更新を有効にする オプションをチェックすることができます。このオプションは、以下の場合にのみ有効です:

- サーバーアプリケーションをビルト オプションがチェックされている。
- 現在のプラットフォームで実行されるクライアントアプリケーションについて、自動更新を有効にする オプションがチェックされている。

この機能を利用するには、[...] ボタンをクリックして、アップデートに使用するファイルのディスク上の場所を指定する必要があります。選択するファイルは、現在のサーバープラットフォームによって異なります:

現在のサーバープラットフォーム	必要なファイル	詳細
macOS	Windows用4D Volume Desktop または Windows クライアントアップデーター アーカイブ	デフォルトでは、Windows用の 4D Volume Desktop アプリケーションを選択します。前もって Windows 上で構築された .4darchive ファイルを選択するには、Shift を押しながら [...]
Windows	macOS クライアントアップデーター アーカイブ	前もって macOS でビルトされた署名入り .4darchive ファイルを選択します。

[クライアントアプリケーションをビルト](#) と [自動更新を有効にする](#) オプションのみを選択することで、サーバーとは異なるプラットフォーム上で .4darchive ファイルをビルトすることができます。

更新通知の表示

サーバーアプリケーションが更新されると、クライアントアプリケーションへの更新通知が自動でおこなわれます。

これは次のように動作します: クライアント/サーバーアプリケーションの新しいバージョンをビルトする際、新しいクライアントは ApplicationName Server フォルダー内の Upgrade4DClient サブフォルダーに圧縮して格納されます (macOS では、これらのフォルダーはサーバーパッケージ内に配置されます)。クロスプラットフォームのクライアントアプリケーションを生成した場合には、各プラットフォーム用に .4darchive という更新ファイルが格納されます:

クライアントアプリケーションに更新を通知するには、古いサーバーアプリケーションを新しいバージョンで置き換えて起動します。あの処理は自動でおこなわれます。

古いバージョンのクライアントが、更新されたサーバーに接続を試みると、新しいバージョンが利用可能である旨を伝えるダイアログがクライアントマシン上に表示されます。ユーザーはバージョンを更新するか、ダイアログをキャンセルできます。

- ユーザーが OK をクリックすると、新バージョンがネットワーク越しにクライアントマシンにダウンロードされます。ダウンロードが完了すると古いクライアントアプリケーションが閉じられて、新しいバージョンが起動しサーバーに接続します。古いバージョンのアプリケーションはマシンのゴミ箱に移動されます。
- ユーザーが キャンセル をクリックすると、更新手続きはキャンセルされます。古いクライアントのバージョンがサーバーの許可する範囲外であれば (後述参照)、アプリケーションは閉じられて、接続することはできません。そうでなければ (デフォルトで) 接続がおこなわれます。

自動更新の強制

更新のダウンロードをキャンセルさせたくない場合、たとえば新しいメジャーバージョンの 4D Server を使用するような場合、新しいバージョンのクライアントアプリケーションを各クライアントマシンに必ずインストールしなければなりません。

更新を強制するには、サーバーアプリケーションと互換性のあるバージョン番号の範囲からクライアントアプリケーションの現バージョン番号を除外します。すると、未更新クライアントからの接続は更新メカニズムによって拒否されます。たとえば、クライアントサーバーアプリケーションの新しいバージョン番号がの 6

の場合、バージョン番号が 5 以下のクライアントアプリケーションを許可しないようにできます。

[現在のバージョン番号](#) はアプリケーションビルトダイアログのクライアント/サーバーページで設定できます。接続を許可するバージョン番号の範囲は [XML キー](#) で設定します。

エラーが発生する場合

クライアントアプリケーションの更新を実行できなかった場合、クライアントマシンには次のメッセージが表示されます："クライアントアプリケーションの更新に失敗しました。アプリケーションは終了します。"

このエラーが発生する原因は複数あります。このエラーが表示されるような場合は、まず次の点をチェックしてみてください:

- パス名 - アプリケーションビルトダイアログや XML キー (たとえば *ClientMacFolderToWin*) で指定されたパス名の有効性をチェックしてください。とくに 4D Volume Desktop へのパスをチェックしてください。
- 読み書き権限 - クライアントマシン上でカレントユーザーがクライアントアプリケーションを更新する書き込みアクセス権を持っているか確認してください。

生成されるファイル

クライアント/サーバーアプリケーションをビルドすると、保存先フォルダー内に Client Server executable という名前の新しいフォルダーが作成されます。このフォルダーにはさらに2つのサブフォルダー、*<ApplicationName> Client* と *<ApplicationName> Server* があります。

エラーが発生した場合これらのフォルダーは作成されません。そのような場合には、エラーの原因を特定するために [ログファイル](#) の内容を確認してください。

<ApplicationName> Client フォルダーは、アプリケーションビルダーを実行したプラットフォームに対応するクライアントアプリケーションを格納します。このフォルダーを各クライアントにインストールします。 *<ApplicationName> Server* フォルダーはサーバーアプリケーションを格納します。

これらのフォルダーの内容はカレントのプラットフォームにより異なります:

- Windows - 各フォルダーに *<ApplicationName>Client.exe* (クライアント用) あるいは *<ApplicationName>Server.exe* (サーバー用) という名前の実行ファイル、およびそれに対応する.rsrファイルが作成されます。これらのフォルダーには、アプリケーション実行のために必要な様々なファイルやフォルダー、および元の 4D Server や 4D Volume Desktop に追加されたカスタマイズ項目も格納されます。
- macOS - 各フォルダーは *<ApplicationName> Client* (クライアント用) と *<ApplicationName> Server* (サーバー用) という名前のアプリケーションパッケージになっています。各パッケージには動作に必要なすべてのファイルが含まれます。macOS では、アプリケーションを実行するためにパッケージをダブルクリックします。

ビルドされた macOS パッケージには、Windows 版のサブフォルダーと同じものが格納されています。ビルドされた macOS パッケージの内容を表示するにはアイコンを Control+クリック して、"パッケージの内容を表示"を選択します。

"クライアントの自動更新を有効にする" オプションを選択している場合、*<ApplicationName>Server* フォルダー/パッケージには追加で *Upgrade4DClient* サブフォルダーが作成されます。このサブフォルダーには macOS/Windows 版のクライアントアプリケーションが圧縮されて格納されます。クライアントアプリケーションを自動更新するときに、このファイルは使用されます。

Webファイルの場所

サーバーやクライアントを Web サーバーとして使用する場合、Web サーバーが使用するファイルを特定の場所に配置しなければなりません:

- *cert.pem* と *key.pem* ファイル (オプション): これらのファイルは TLS 接続とデータ暗号化コマンドに使用されます。
- デフォルト Web ルートフォルダー (WebFolder)

インストール場所:

- Windows
 - サーバーアプリケーション - *Client Server executable/<ApplicationName>Server/Server Database* サブフォルダー内にこれらの項目を配置します。
 - クライアントアプリケーション - *Client Server executable/<ApplicationName>Client* サブフォルダー内にこれらの項目を配置します。
- macOS
 - サーバーアプリケーション - *<ApplicationName>Server* ソフトウェアパッケージと同階層にこれらの項目を配置します。
 - クライアントアプリケーション - *<ApplicationName>Client* ソフトウェアパッケージと同階層にこれらの項目を配置します。

シングルユーザークライアントアプリケーションの埋め込み

4D ではクライアントアプリケーションにコンパイル済ストラクチャーを埋め込むことができます。この機能を使用すると、たとえば、`.4dlink` ファイルを `OPEN DATABASE` コマンドで実行することで異なるサーバーアプリケーションにアクセスできるような "ポータル" アプリケーションをユーザーに提供することができます。

この機能を有効化するためには、`buildApp` 設定ファイルに `DatabaseToEmbedInClientWinFolder` または `DatabaseToEmbedInClientMacFolder` キーを追加します。いずれかのキーが存在する場合、アプリケーションビルドプロセスの途中で組み込みシングルユーザー-applicationが生成され、コンパイルされたストラクチャーが (`EnginedServer.4Dlink` ファイルの代わりに) "Database" フォルダー内に置かれます。

- シングルユーザー-application内に "Default Data" フォルダーがあれば、アプリケーションにはライセンスが埋め込まれます。
- シングルユーザー-application内に "Default Data" フォルダーがなければ、データファイルおよびライセンスなしでアプリケーションが実行されます。

基本シナリオは以下の通りです：

- アプリケーションビルド ダイアログボックス内にて、"コンパイルされたストラクチャーをビルド" オプションを選択し、シングルユーザー-modeで使用されるアプリケーションの `.4DC` または `.4DZ` ファイルを生成します。
- クライアント/サーバーアプリケーションの `buildApp.4DSettings` ファイル内で、コンパイルされたシングルユーザー-applicationを格納しているフォルダへのパスを以下の xml キーに指示します：
 - `DatabaseToEmbedInClientWinFolder`
 - `DatabaseToEmbedInClientMacFolder`
- クライアント/サーバーアプリケーションをビルドします。これは以下のように動作します：
 - シングルユーザー-applicationのフォルダー全体が、組み込みクライアントの "Database" フォルダー内にコピーされます。
 - "Database" フォルダーの `EnginedServer.4Dlink` ファイルは生成されません。
 - シングルユーザー-applicationのコピーが持つ `.4DC`、`.4DZ`、`.4DIndy` ファイルは、組み込みクライアントの名前へとファイル名が変更されます。
 - `PublishName` キーは、組み込みクライアントの `info.plist` にコピーされません。
 - シングルユーザーデータベースに "Default Data" フォルダーがない場合、組み込みクライアントはデータなしで実行されます。

4D Server の自動アップデート機能（[現在のバージョン](#) 番号、`SET UPDATE FOLDER` コマンドなど...）は、シングルユーザー-applicationにおいても標準のリモートアプリケーションと同様に動きます。接続時、シングルユーザー-applicationは `<code>CurrentVers` キーを 4D Server バージョンレンジと比較します。レンジ外だった場合、アップデートされているシングルユーザー-applicationがサーバーからダウンロードされ、アップデーターがローカルアップデートプロセスを実行します。

クライアントおよびサーバーキャッシュフォルダー名のカスタマイズ

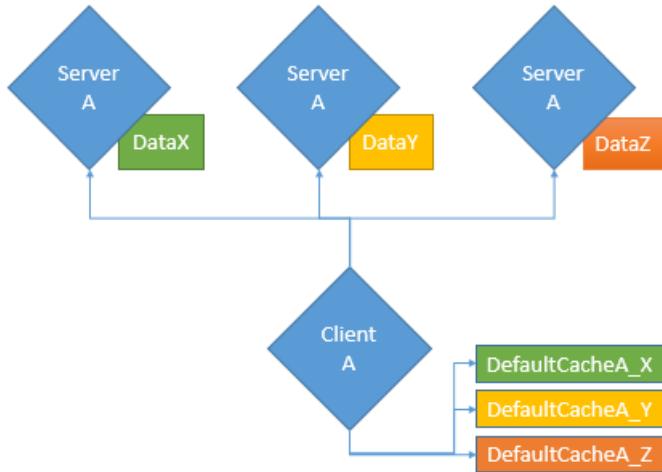
クライアントおよびサーバーのキャッシュフォルダーは、リソースやコンポーネントなどの共有要素を格納するのにしようされます。これらは、サーバーとリモートクライアント間の通信を管理するのに必要です。クライアント/サーバーアプリケーションは、クライアントおよびサーバーシステム両方のキャッシュフォルダーにデフォルトパス名を使用します。

特殊な場合においては、特定のアーキテクチャーを実装するために、これらのフォルダー名をカスタマイズする必要があるかもしれません（後述参照）。このため、4D は `buildApp` 設定ファイルにて使用可能な `ClientServerSystemFolderName` および `ServerStructureFolderName` キーを提供しています。

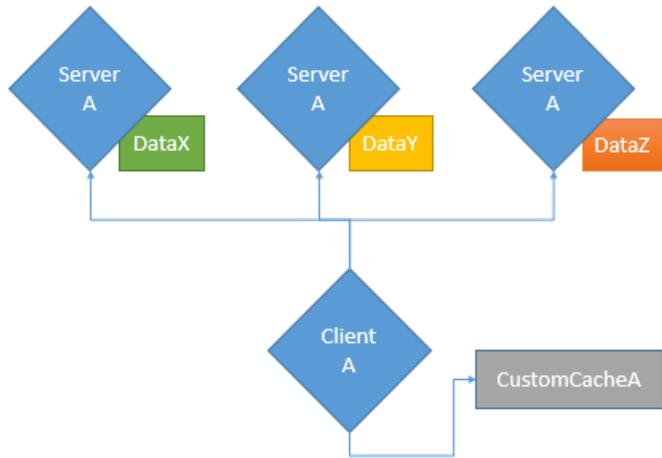
クライアントキャッシュフォルダー

それぞれ異なるデータセットを使用している以外は同様の複数の組み込みサーバーにクライアントアプリケーションが接続するような場合、クライアント側のキャッシュフォルダー名をカスタマイズすると便利かもしれません。このような場合に、同じローカルリソースを複数回ダウンロードするのを避けるため、同一のローカルキャッシュフォルダーを使用することができます。

- デフォルトの設定（サーバーへの接続ごとに専用のキャッシュフォルダーがダウンロード/更新されます）：



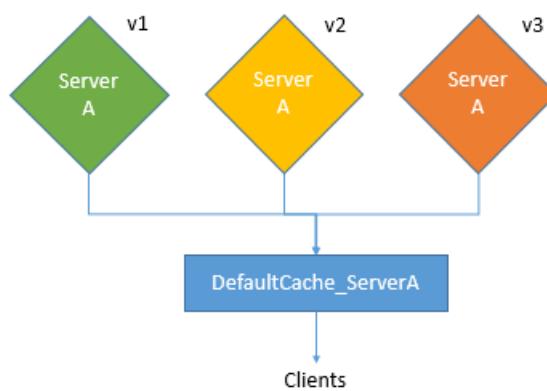
- `ClientServerSystemFolderName` キーの使用 (すべてのサーバーに対して同じキャッシュフォルダーが使用されます):



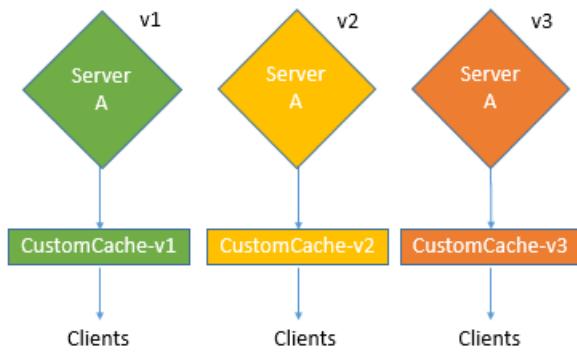
サーバーキャッシュフォルダー

それぞれ異なる 4Dのバージョンでビルドされた同じサーバーアプリケーションを同一のマシン上で実行する場合、サーバー側のキャッシュフォルダ名をカスタマイズすると便利です。各サーバーで独自のリソースを使用するには、サーバーキャッシュフォルダーをカスタマイズする必要があります。

- デフォルトの設定 (同じサーバーアプリケーションは同じキャッシュフォルダーを共有します):



- `ServerStructureFolderName` キーの使用 (各サーバーアプリケーションに専用のキャッシュフォルダーが使用されます):



プラグイン & コンポーネントページ

このページでは、シングルユーザーまたはクライアント/サーバーアプリケーションに含める [プラグイン](#) や [コンポーネント](#)、[モジュール](#) を設定できます。

このページには、現在の 4D アプリケーションにロードされている要素がリストされます:

- アクティブ 列 - ビルドするアプリケーションパッケージに項目を統合するかどうかを指定します。デフォルトですべての項目が選択されています。プラグインやコンポーネント、モジュールを除外するには、チェックボックスの選択を外します。
 - プラグイン&コンポーネント 列 - プラグイン/コンポーネント/モジュールの名称を表示します。
 - ID 列 - 要素の ID (あれば) を表示します。
 - タイプ 列 - その要素がプラグイン・コンポーネント・モジュールのいずれであるかが表示されます。

プラグインやコンポーネントの追加

その他のプラグインやコンポーネントをアプリケーションに統合したい場合には、4D Server や 4D Volume Desktop の Plugins や Components フォルダーにそれらを配置します。ソースアプリケーションのフォルダーから内容をコピーするメカニズム（[4D Volume Desktop フォルダーのカスタマイズ](#) 参照）

照) により、どんなタイプのファイルでもアプリケーションに統合することができます。

同じプラグインの異なるバージョンが見つかった場合(現在 4D にロードされているものと同じプラグインが、ソースアプリケーションのフォルダーにも配置されている場合など)、4D Volume Desktop/4D Server フォルダーにインストールされているバージョンが優先されます。他方、同じコンポーネントが両方にインストールされていた場合は、アプリケーションを開くことはできません。

配布するアプリケーションでプラグインやコンポーネントを使用するには、それぞれ適切なライセンスが必要な場合があります。

モジュールの選択解除

モジュールとは、特定の機能を制御するために 4D で使用される組み込みのコードライブラリです。ビルトするアプリケーションがモジュールの機能を使用しないことが分かっている場合、アプリケーションのファイルサイズを小さくするために、そのモジュールの選択をリストで解除することができます。

警告: モジュールの選択を解除すると、ビルトしたアプリケーションが期待通りに動作しなくなる可能性があります。特定のモジュールが、アプリケーションから呼び出されることがないことが確実でない場合は、選択したままにしておくことが推奨されます。

以下の任意モジュールは、選択を解除することができます:

- CEF: Chromium Embedded Framework ライブリ。埋め込み Web レンダリングエンジンを使用する [Web エリア](#) や、[4D View Pro エリア](#) を実行する際に必要です。CEF が選択されていない状態でこれらのエリアを呼び出すと、空白が表示されたり、エラーが発生したりします。
- MeCab: 日本語のテキストインデックスに使用されるライブラリです ([MeCab のサポート \(日本語版\)](#) 参照)。このモジュールの選択を解除すると、テキストインデックスが日本語で再構築されます。

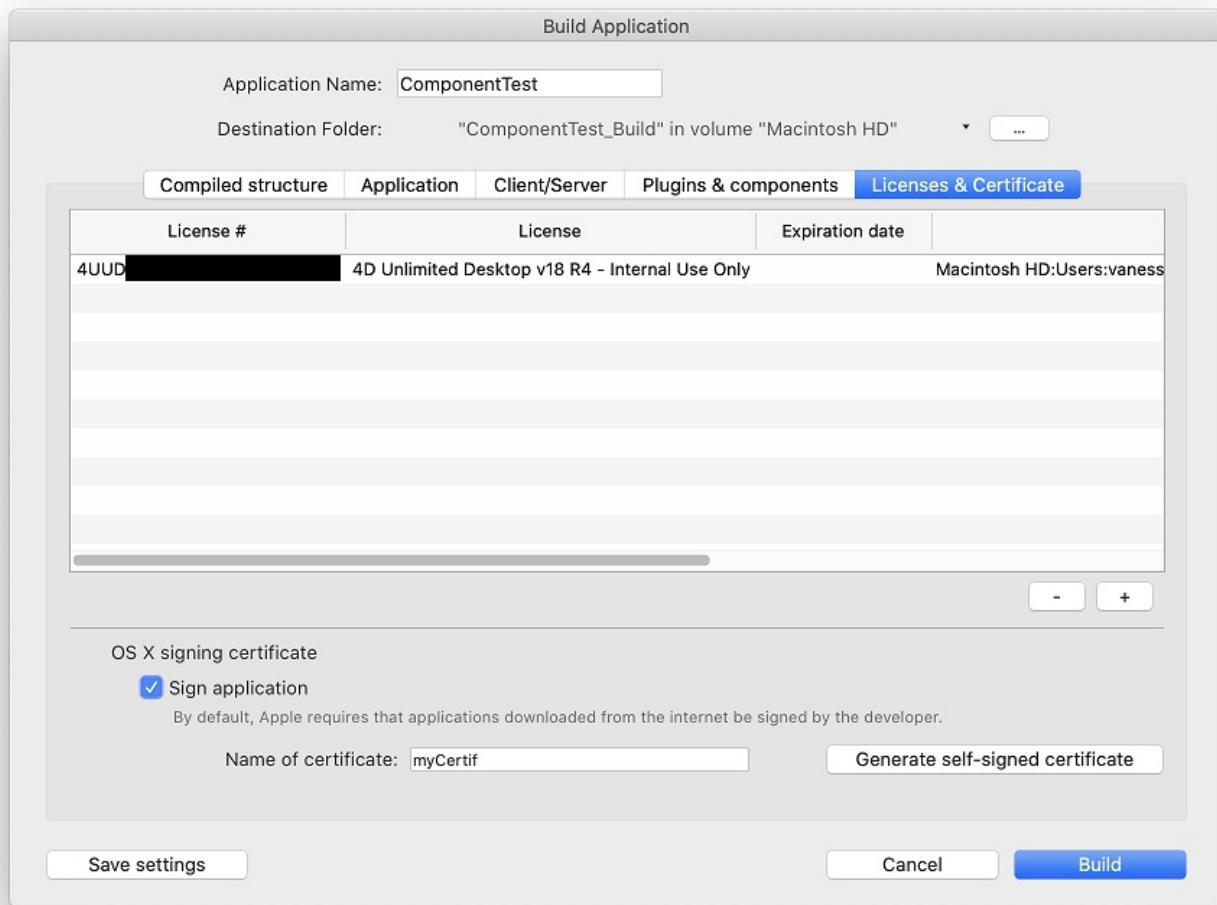
異なるプラットフォームで使用する日本語のアプリケーションで MeCab の選択を解除する場合、必ず、クライアント/サーバーのビルトと [クライアントアプリケーションをビルト](#) (実行中の OS 用) の両方で選択を解除しないと、アプリケーションに大きな不具合が生じます。

- PHP: PHP の機能とコマンドを 4D で使用するために必要です (設定の [PHP ページ](#) 参照)。
- SpellChecker: 入力エリアと 4D Write Pro エリアで利用可能なビルトインの [スペルチェック機能](#) とコマンドに使用されます。
- 4D Updater: クライアントの [自動更新](#) をコントロールし、[サーバーの自動更新](#) のための SET UPDATE FOLDER コマンドで使用されます。

ライセンス & 証明書ページ

ライセンス&証明書のページでは、次のようなことができます:

- シングルユーザーのスタンドアロンアプリケーションに統合するライセンス番号を指定します。
- macOS 環境下では、証明書を使用してアプリケーションに署名することができます。



ライセンスリスト

アプリケーションに統合するのに使用できる配布ライセンスの一覧を表示します。デフォルトでリストは空です。アプリケーションをビルドするには *4D Developer Professional* ライセンスと、その開発ライセンスに対応する *4D Desktop Volume* ライセンスを指定しなければなりません。現在使用しているものとは別の *4D Developer Professional* ライセンス（およびその付属ライセンス）を追加することもできます。

ライセンスを追加または取り除くにはウィンドウ下部の [+] または [-] ボタンをクリックします。

[+] ボタンをクリックすると、ファイルを開くダイアログが表示され、マシンの *Licenses* フォルダーの内容が表示されます。このフォルダーの場所については詳しくは [Get 4D folder](#) コマンドの説明を参照してください。

開発ライセンスとそれに対応した配布ライセンスを選択します。これらのファイルは *4D Developer Professional* ライセンスや *4D Desktop Volume* ライセンスをアクティベーションした際、この場所にコピーされます。

ファイルを選択すると、リストに選択内容が反映されます：

- ライセンス # - 製品ライセンス番号
- ライセンス - プロダクト名
- 有効期限 - ライセンスの有効期限（あれば）
- パス - ディスク上のライセンスの場所

ライセンスが有効でない場合、警告が表示されます。

必要なだけ有効なファイルを選択することができます。実行可能アプリケーションをビルドする際に、4D は最も適切なライセンスを使用します。

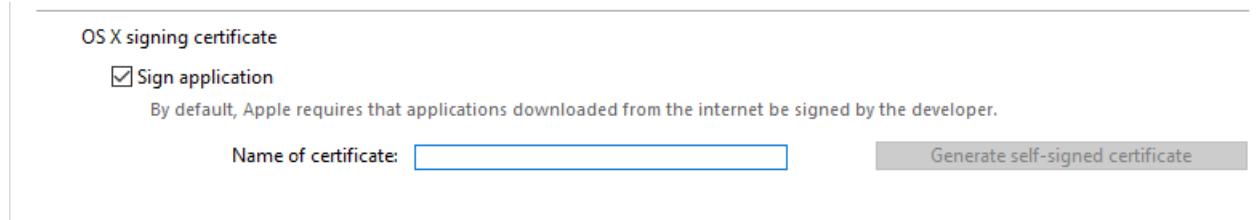
"R-リリース" バージョンのアプリケーションをビルドするには、専用の "R" ライセンスが必要です ("R" 製品用のライセンス番号は "R-" から始まる番号です)。

アプリケーションビルド後、配布ライセンスファイルは実行可能ファイルと同階層 (Windows) やパッケージ内 (macOS) に自動でコピーされます。

OS X 署名に使用する証明書

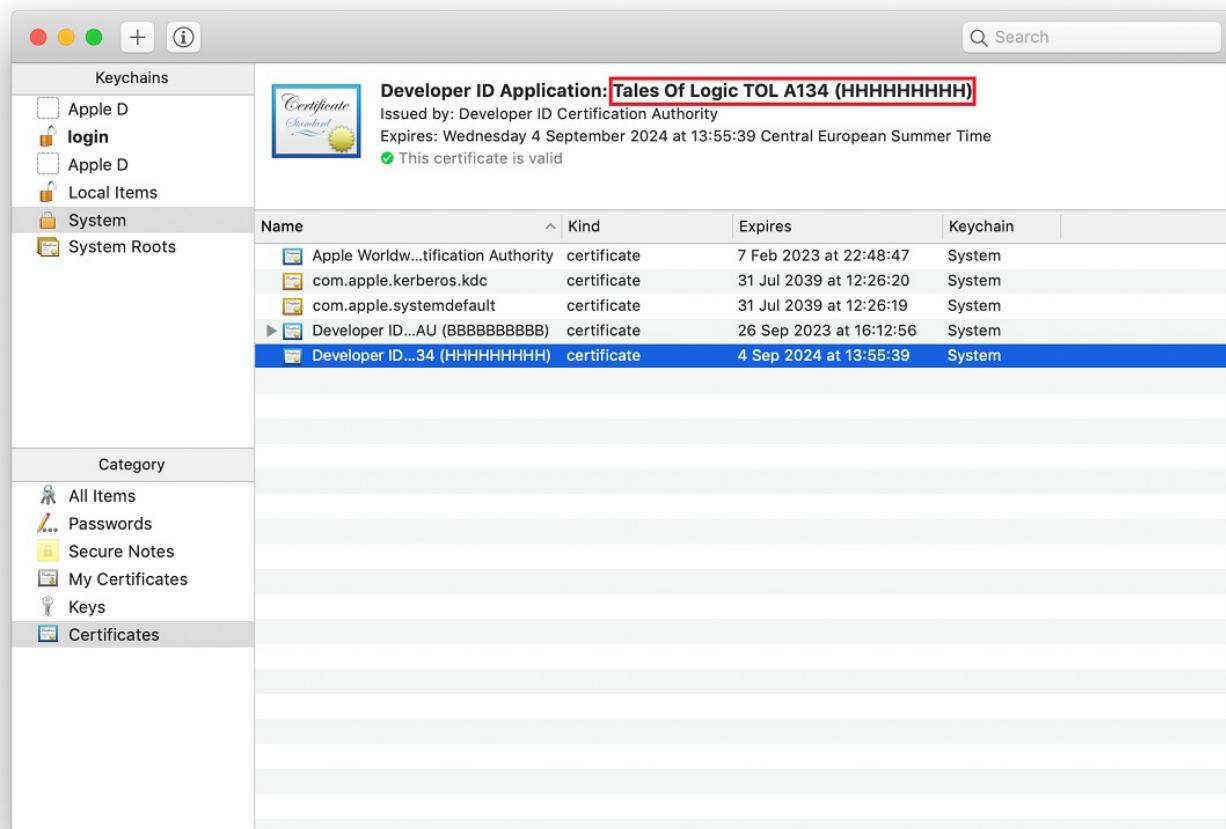
アプリケーションビルダーは、macOS 環境下において組み込み4Dアプリに署名をする機能を備えています (macOS のシングルユーザーアプリ、コンポーネント、サーバーおよびクライアントアプリ)。アプリケーションを署名することにより、macOSにおいて「Mac App Store と確認済みの開発元からのアプリケーションを許可」のオプションが選択されているときに Gatekeeper の機能を使用してアプリケーションを実行することが可能になります (後述の "Gatekeeperについて" を参照ください)。

- アプリケーションに署名 オプションにチェックをすると、macOS のアプリケーションビルト処理に認証が含まれます。4D はビルドの際に、認証に必要な要素の有無をチェックします:



このオプションは Windows と macOS 両方の環境で表示されますが、macOS の場合においてのみ有効です。

- 認証名 - Apple によって有効化されたデベロッパー認証名を入力してください。この認証名は通常、キーチェーンアクセスユーティリティ内の証明書の名前と一緒に表示されます (下図の赤枠):



Apple からデベロッパー認証を取得するためには、キーチェーンアクセスのメニューのコマンドを使用するか、次のリンクへ移動してください:

<http://developer.apple.com/library/mac/#documentation/Security/Conceptual/CodeSigningGuide/Procedures/Procedures.html>。

この証明書の取得には Apple の codesign ユーティリティが必要になります。このユーティリティはデフォルトで提供されており、通常 "/usr/bin/" フォルダーにあります。エラーが起きた際には、このユーティリティがディスク上にあるかどうかを確認してください。

- 自己署名証明書の生成 - 自己署名証明書を生成するための "証明書アシスタント" を実行します。Apple 社のデベロッパー認証を持たない場合には、自己署名証明書を提供する必要があります。この証明書を使うと、アプリケーションを内部的に運用する場合に警告が表示されません。アプリケーションを外部で運用する場合 (http やメールを介した場合) には、アプリケーションの開発者が不明であるという警告が macOS での起動時に表示されます。その場合でもユーザーはアプリケーションを "強制的" に起動することができます。

"証明書アシスタント" では、オプションを選択します：



Apple Developer Program に加入し、アプリケーションの公証（後述参照）に必要なデベロッパー認証を取得することが推奨されます。

Gatekeeper について

Gatekeeper とは macOS のセキュリティ機能で、インターネットからダウンロードしてきたアプリケーションの実行を管理するものです。もしダウンロードしたアプリケーションが Apple Store からダウンロードしたものではない、または署名されていない場合には実行が拒否されます。

Apple Silicon マシンでは、4D コンポーネント は実際に署名されている必要があります。署名されていないコンポーネントの場合、アプリケーション起動時にエラー ("lib4d-arm64.dylib を開けません...")

ビルトボタンをクリックすると、4D は 保存先フォルダー に Final Application フォルダーを作成し、その中に指定したアプリケーション名のサブフォルダーを作成します。

ノータリゼーション（公証）について

macOS 10.14.5 (Mojave) および 10.15 (Catalina)において、アプリケーションのノータリゼーション（公証）が Apple より強く推奨されています。

す。公証を得ていないアプリケーションをインターネットからダウンロードした場合、デフォルトでブロックされます。

Apple の公証サービスを利用するのに必要な条件を満たすため、4D の [ビルトインの署名機能](#) が適合されました。公証自体はデベロッパーによっておこなわなくてはいけないもので、4D とは直接関係ありません。なお、Xcode のインストールが必須である点に注意してください。公証についての詳細は [4D ブログ記事 \(英語\)](#) や関連の [テクニカルノート \(日本語\)](#) を参照ください。

公証についての詳細は、

[Apple のデベロッパー Web サイト](#) を参照ください。

アプリケーションアイコンのカスタマイズ

4Dは、ダブルクリックで実行可能なアプリケーションにデフォルトアイコンを割り当てますが、アプリケーションごとにこのアイコンをカスタマイズできます。

- macOS - アプリケーションビルドの際にアイコンをカスタマイズするには、icns タイプのアイコンファイルを作成し、それを Project フォルダーと同階層に配置しておきます。

Apple, Inc. より、icns アイコンファイルを作成するツールが提供されています。(詳細については、[Apple documentation](#) を参照してください。)

アイコンファイルの名前は、プロジェクトファイル名 + "*.icns*" 拡張子でなければなりません。4D は自動でこのファイルを認識し、

- Windows - アプリケーションビルドの際にアイコンをカスタマイズするには、.ico タイプのアイコンファイルを作成し、それを Project フォルダーと同階層に配置しておきます。

アイコンファイルの名前は、プロジェクトファイル名 + ".ico" 拡張子でなければなりません。4Dは自動でこのファイルを認識し、アイコンとして使用します。

また、buildApp.4DSettings ファイルにて、使用すべきアイコンを [XML keys](#) (SourcesFiles の項参照)によって指定することも可能です。次のキーが利用できます:

- RuntimeVLIconWinPath
- RuntimeVLIconMacPath
- ServerIconWinPath
- ServerIconMacPath
- ClientMacIconForMacPath
- ClientWinIconForMacPath
- ClientMacIconForWinPath
- ClientWinIconForWinPath

データファイルの管理

データファイルを開く

ユーザーが組み込みアプリ、またはアプリのアップデート (シングルユーザー、またはクライアント/サーバーアプリ) を起動すると、4D は有効なデータファイルを選択しようとします。アプリケーションによって、複数の場所が順次検索されます。

組み込みアプリ起動時のオープニングシーケンスは以下になっています:

- 4D は最後に開かれたデータファイルを開こうとします。詳しくは [後述の説明](#) を参照ください (これは初回起動時には適用されません)。
- 見つからない場合、4D は .4DZ ファイルと同階層の Default Data フォルダー内にあるデータファイルを、読み取り専用モードで開こうとします。
- これも見つからない場合、4D は標準のデフォルトデータファイルを開こうとします (.4DZ ファイルと同じ場所にある、同じ名前のファイル)。
- これも見つからない場合、4D は "データファイルを開く" ダイアログボックスを表示します。

最後に開かれたデータファイル

最後に開かれたファイルへのパス

4D でビルトされたスタンドアロンまたはサーバーアプリケーションは、最後に開かれたデータファイルのパスをアプリケーションのユーザー設定フォルダー内に保存します。

アプリケーションのユーザー設定フォルダーの場所は、以下のコマンドで返されるパスに対応しています:

```
userPrefs:=Get 4D folder(Active 4D Folder)
```

データファイルパスは *lastDataPath.xml* という名前の専用ファイルに保存されます。

これにより、アプリケーションのアップデートを提供したときにも、ローカルのユーザーデータファイル（最後に使用されたデータファイル）が初回の起動から自動的に開かれます。

このメカニズムは標準的な運用に適しています。しかしながら特定の場合、たとえば組み込みアプリケーションを複製した場合などにおいて、データファイルとアプリケーションのリンクを変えたいことがあるかもしれません（次章参照）。

データリンクモードの設定

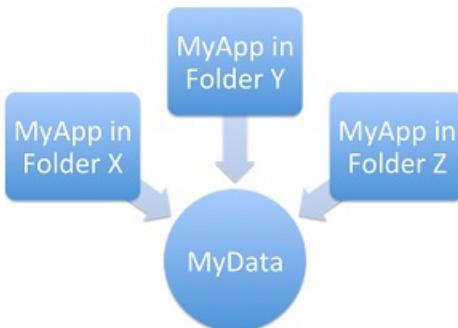
コンパイルされたアプリケーションでは、4D は最後に使用されたデータファイルを自動的に使用します。デフォルトでは、データファイルのパスはアプリケーションのユーザー設定フォルダー内に保存され、**アプリケーション名** でリンクされます。

異なるデータファイルを使用するために組み込みアプリを複製する場合には、この方法は適さないかもしれません。複製されたアプリは同じアプリケーションユーザー設定フォルダーを共有するため、同じデータファイルを使用することになります（最後に使用されたファイルが開かれるため、データファイル名を変更した場合でも結果は同じです）。

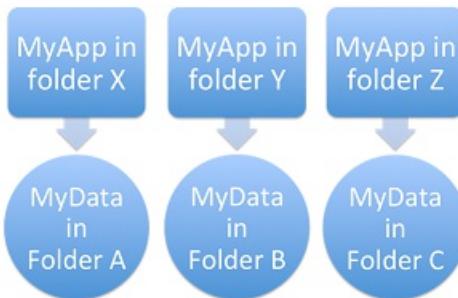
そのため 4D では、アプリケーションパスを使用してデータファイルとリンクすることも可能です。このとき、データファイルは特定のパスを使用してリンクされるので、最後に開かれたファイルであるかは問われません。この設定を使うには、データリンクモードの基準を アプリケーションパス に設定します。

このモードを使えば、組み込みアプリがいくつあっても、それぞれが専用のデータファイルを使えます。ただし、デメリットもあります：アプリケーションパッケージを移動させてしまうとアプリケーションパスが変わってしまうため、データファイルを見つけられなくなります。この場合、ユーザーは開くデータファイルを指定するダイアログを提示され、正しいファイルを選択しなくてはなりません。一度選択されれば、*lastDataPath.xml* ファイルが更新され、新しい "executablePath" 属性のエントリーが保存されます。

データがアプリケーション名でリンクされている場合の複製：



データがアプリケーションパスでリンクされている場合の複製：



このデータリンクモードはアプリケーションビルトの際に選択することができます。オブジェクトにヘルプメッセージを関連付けるには：

- アプリケーションビルトの [アプリケーションページ](#) または [クライアント/サーバーページ](#) を使用する。
- シングルユーザーまたはサーバーアプリケーションの LastDataPathLookup XMLキーを使用する。

デフォルトのデータフォルダーを定義する

4D では、アプリケーションビルト時にデフォルトのデータファイルを指定することができます。アプリケーションの初回起動時に、開くべきローカルデータファイルが見つからなかった場合（前述の [オープニングシーケンス](#) 参照）、デフォルトのデータファイルが読み取り専用モードで自動的に開かれます。この機能を

使って、組み込みアプリを初回起動したときのデータファイル作成・選択の操作をより制御することができます。

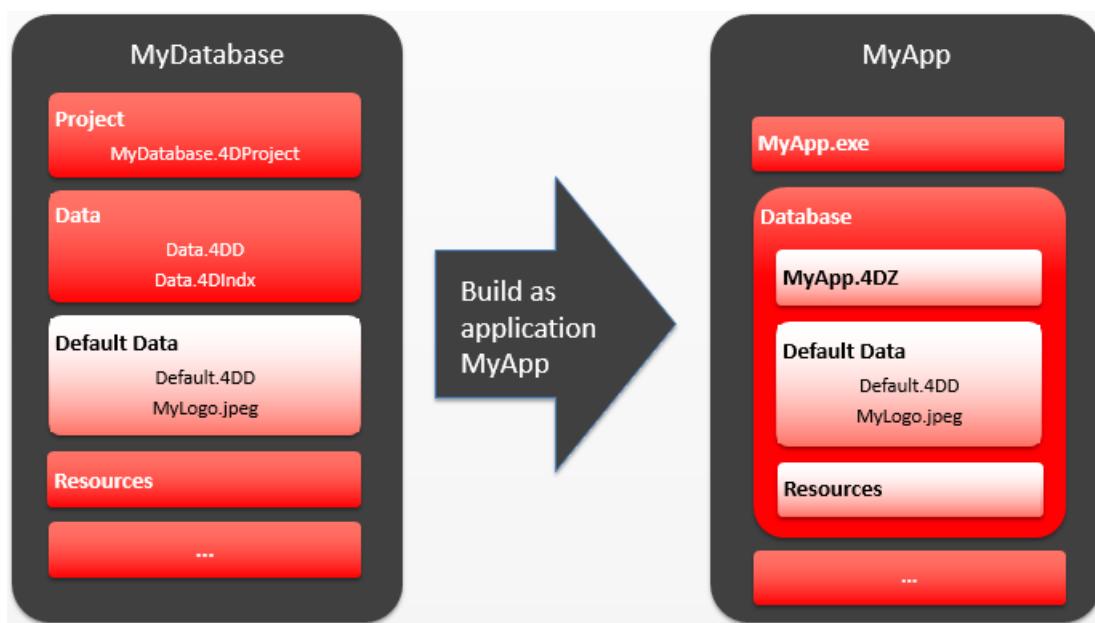
具体的には、次のような場合に対応できます:

- 新しい、またはアップデートされた組み込みアプリを起動したときに、"データファイルを開く" ダイアログが表示されるのを防ぐことができます。たとえば、デフォルトデータファイルが開かれたことを起動時に検知して、独自のコードやダイアログを実行して、ローカルデータファイルの作成や選択を促すことができます。
- デモアプリなどの用途で、読み取り専用データしか持たない組み込みアプリを配布することができます。

デフォルトのデータファイルを定義・使用するには:

- デフォルトのデータファイル（名称は必ず "Default.4DD"）を、データベースプロジェクトのデフォルトフォルダー（名称は必ず "Default Data"）内に保存します。このデフォルトのデータファイルには、プロジェクト構成に応じて必要なファイルもすべて揃っている必要があります：インデックス (.4DIndx)、外部BLOB、ジャーナル、他。必ず、有効なデフォルトデータファイルを用意するようにしてください。なお、デフォルトデータファイルはつねに読み取り専用モードで開かれため、データファイルの作成前に、あらかじめ大元のストラクチャー設定の "ログを使用" オプションを非選択にしておくことが推奨されます。
- アプリケーションをビルドすると、このデフォルトデータフォルダーが組み込みアプリに統合されます。同フォルダー内ファイルはすべて一緒に埋め込まれます。

この機能を図示すると次のようになります:



デフォルトのデータファイルが初回起動時に検知された場合、データファイルは自動的に読み取り専用モードで開かれ、データファイルの変更を伴わないカスタムオペレーションを実行できるようになります。

クライアント接続の管理

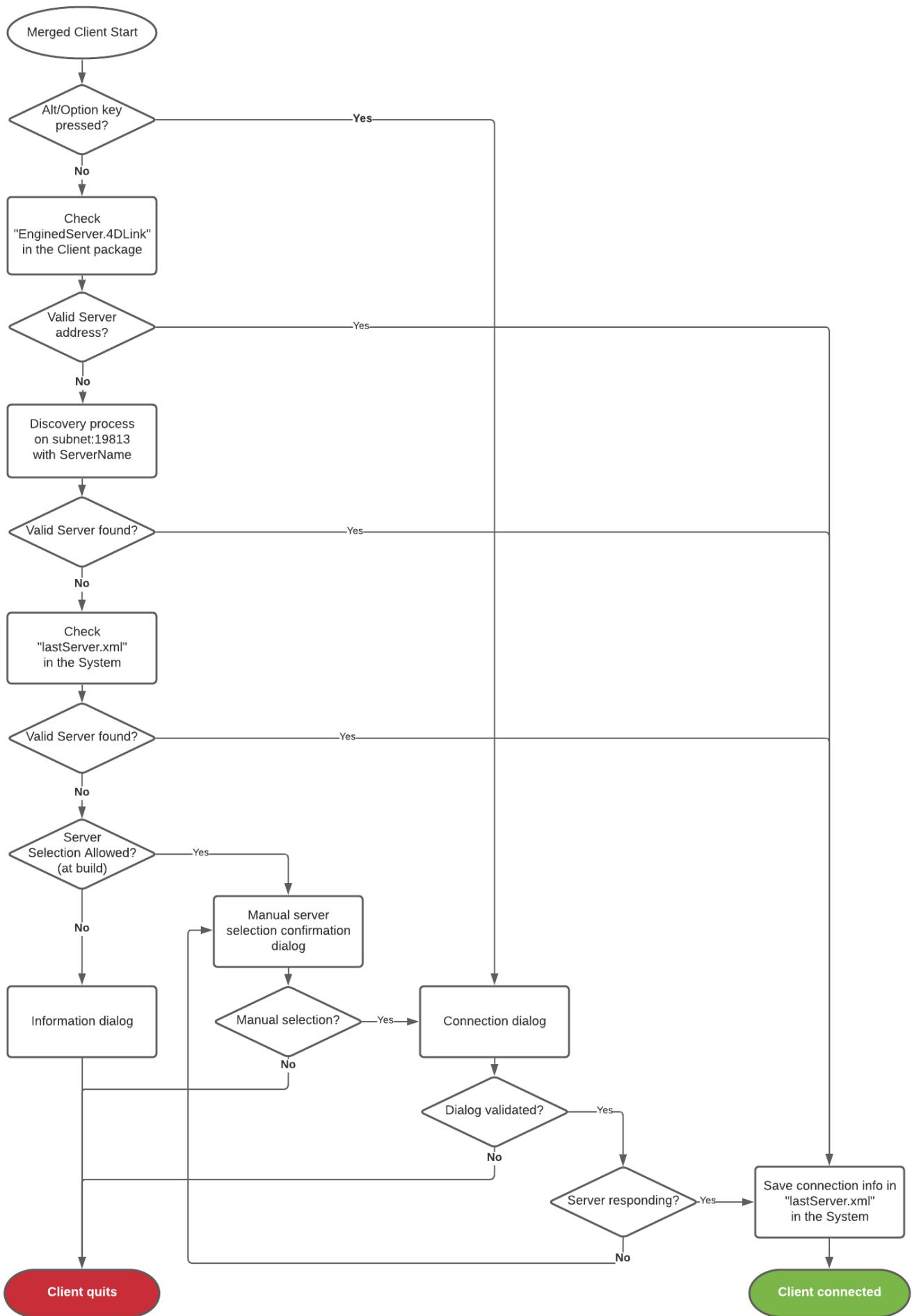
ここでは、組み込みクライアントアプリが運用環境において対象サーバーへと接続する際のメカニズムについて説明します。

接続シナリオ

組み込みクライアントアプリの接続プロセッサーは、専用サーバーが使用不可能な場合にも柔軟に対応します。4Dクライアントアプリのスタートアップシナリオは、次のとおりです：

- クライアントアプリ内の "EnginedServer.4DLink" ファイルに有効な接続情報が保存されていた場合、クライアントアプリは指定されたサーバーアドレスへ接続を試みます。
または
クライアントアプリは検索サービスを使用してサーバーへの接続を試みます（同じサブネット内に公開されたサーバー名に基づいて検索します）。
- これが失敗した場合、クライアントアプリケーションは、アプリケーションのユーザー設定フォルダーに保存されている情報 ("lastServer.xml" ファイル、詳細は後述参照）を使用してサーバーへの接続を試みます。
- これが失敗した場合、クライアントアプリケーションは接続エラーダイアログボックスを表示します。
 - ユーザーが「選択...」ボタンをクリックした場合、標準の "サーバー接続" ダイアログボックスが表示されます（ビルドの段階で許可されていた場合に限ります。詳細は後述）。

- ユーザーが 終了 ボタンをクリックした場合、クライアントアプリケーションは終了します。
4. 接続が成功した場合、クライアントアプリケーションは将来の使用のために、その接続情報をアプリケーションのユーザー設定フォルダーに保存します。
- この手順を図示すると以下のようになります：



最後に使用したサーバーパスを保存する

最後に使用され検証されたサーバーパスは、アプリケーションのユーザー設定フォルダー内の "lastServer.xml" ファイルに自動的に保存されます。この

フォルダーは次の場所に保存されています:

```
userPrefs:=Get 4D folder(Active 4D Folder)
```

このメカニズムは、最初に指定したサーバーが何らかの理由（例えばメンテナスモードなど）で一時的に使用できないケースに対応します。こういった状態が初めて起こったときにはサーバー選択ダイアログボックスが表示され（ただし許可されていた場合に限ります、後述参照）、ほかのサーバーをユーザーが手動で選択すると、その接続が成功した場合にはそのパスが保存されます。それ以降に接続ができなかった場合には、"lastServer.xml" のパス情報によって自動的に対処されます。

- ネットワークの設定などの影響で、クライアントアプリが恒久的に検索サービスを使ったサーバー接続ができない場合には、ビルド時にあらかじめ "BuildApp.4DSettings" ファイル内の **IPAddress** キーでホスト名を指定しておくことが推奨されます。このメカニズムはあくまで一時的な接続不可状態の場合を想定しています。
- スタートアップ時に Alt/Option キーを押しながら起動してサーバー接続ダイアログボックスを表示する方法は、すべての場合において可能です。

エラー時のサーバー選択ダイアログボックス使用の可・不可

組み込みクライアントアプリがサーバーに接続できない場合、標準のサーバー選択ダイアログボックスを表示するかどうかは設定しておくことができます。この設定は、アプリケーションをビルドするマシン上の **ServerSelectionAllowedXML** キーの値によって制御されます:

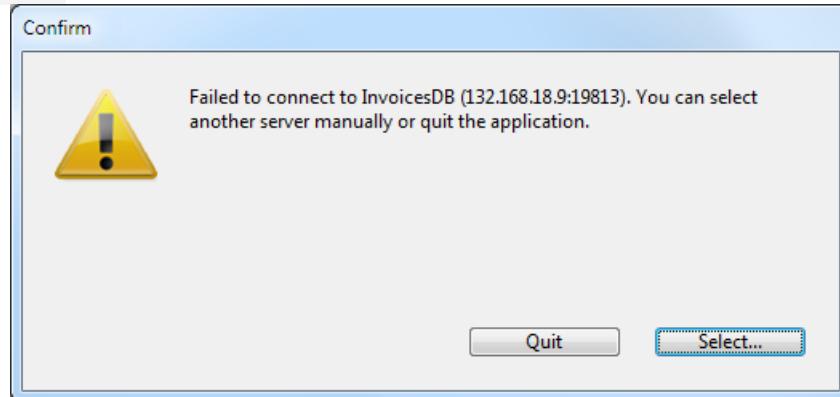
- エラーメッセージを表示し、サーバー選択ダイアログボックスを表示させない。デフォルトの挙動です。アプリケーションは終了する以外の選択肢がありません。

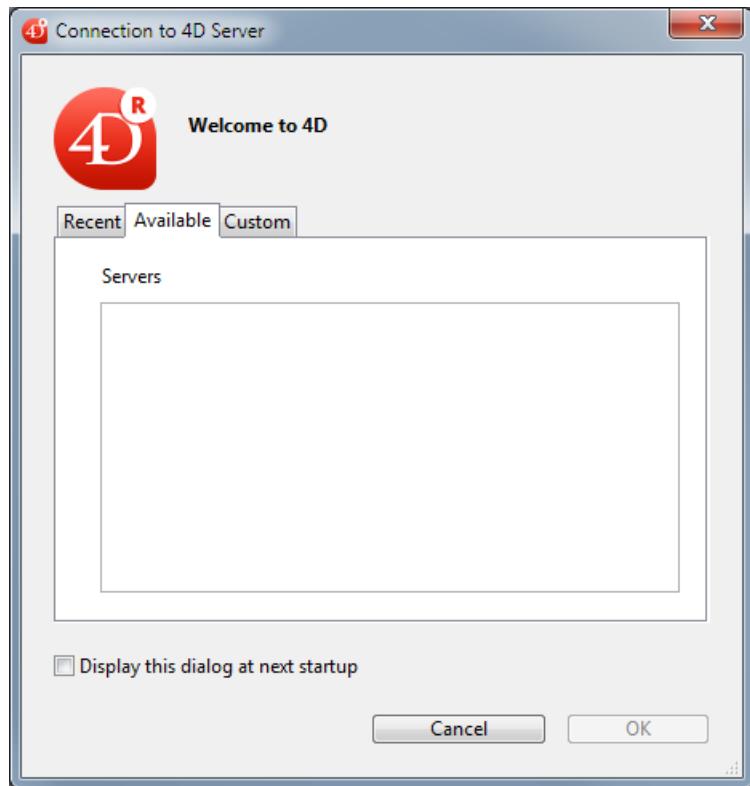
ServerSelectionAllowed : False 値、またはキーを省略



- エラーメッセージを表示し、サーバー選択ダイアログボックスへのアクセスを可能にする。ユーザーは **選択...** ボタンをクリックする事によってサーバー選択ウィンドウにアクセスできます。

ServerSelectionAllowed : True 値





サーバーまたはシングルユーユーアプリケーションの自動更新

以下に、サーバーや組み込みシングルユーユーアプリケーションをアップデートする場合のシナリオを説明します：

この手順は、次のランゲージコマンドを使って大部分を自動化することができます：`SET UPDATE FOLDER`、`RESTART 4D`、そして `Get last update log path` (モニタリング操作用)。これらのコマンドを使い、4Dアプリケーションに、以下に説明する自動更新手順をトリガーする機能を実装します。具体的には、メニュー命令や、バックグラウンドで動作するプロセスなどを実装し、サーバーにアーカイブがあるかどうかを定期的にチェックします。

また、Windows で保護されたファイルを使用できるように、インストール権限を昇格させる XMLキーもあります ([アプリケーションビルト設定ファイル](#) のマニュアルを参照ください)。

原則的に、サーバーアプリケーションや組み込みのシングルユーユーアプリケーションを更新するには、ユーザーの介入（またはカスタムのシステムルーチンのプログラミング）が必要です。

1. HTTPサーバーなどを使用して、サーバーアプリケーションまたは組み込みシングルユーユーアプリケーションの新バージョンを本番環境のマシンに転送します。
2. 本番環境のアプリケーションでは、`SET UPDATE FOLDER` コマンドを呼び出します。このコマンドは、カレントアプリケーションの "保留中" のアップデートが置かれたフォルダーの場所を指定します。任意で、本番環境バージョンのカスタム要素（ユーザーファイル）をこのフォルダーにコピーすることもできます。
3. 本番環境のアプリケーションで、`RESTART 4D` コマンドを呼び出します。このコマンドは、"updater" という名前のユーティリティプログラムの実行を自動的にトリガーします。このユーティリティは カレントのアプリケーションを終了し、"保留中" のアップデートが指定されている場合はそれで置き換え、カレントのデータファイルでアプリケーションを再起動します。旧バージョンは名称変更されます。

この手順は、サービスとして実行される Windows サーバーアプリケーションと互換性があります。

アップデートログ

インストール手順により、ターゲットマシン上の組み込みアプリケーション（クライアント、サーバー、またはシングルユーザー）の更新処理の詳細が記録されたログファイルが作成されます。このファイルは、インストール中に発生したエラーの分析に役立ちます。

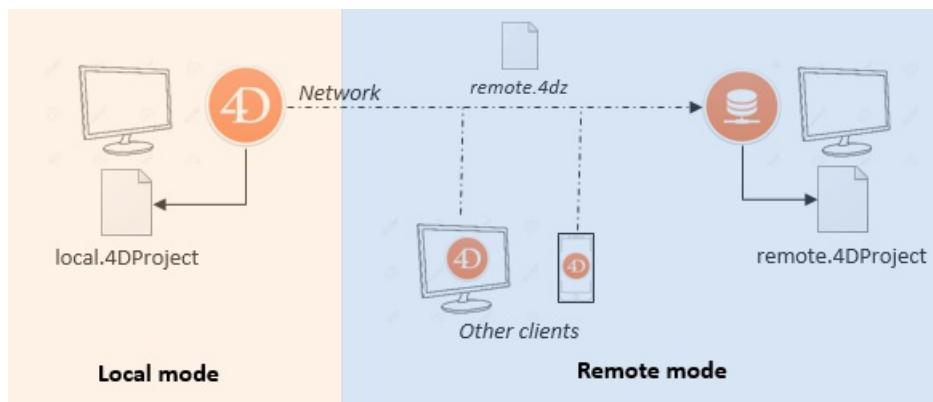
アップデートログは、`YYYY-MM-DD_HH-MM-SS_log_X.txt` という名前で、たとえば、2021年 8月 25日の 14:23 に作成されたファイルに対しては `2021-08-25_14-23-00_log_1.txt` という名前になります。

このファイルは、システムユーザー フォルダー内の "Updater" アプリケーション フォルダー内に作成されます。このファイルの場所は、`Get last update log path` コマンドでいつでも確認することができます。

クライアント/サーバー管理

組み込みクライアント/サーバーアプリケーションまたはリモートプロジェクトの形で、4Dデスクトップアプリケーションをクライアント/サーバー構成で運用することができます。

- 組み込みクライアント/サーバーアプリケーションは [アプリケーションビルダー](#) を使って生成します。これらは、アプリケーションの運用に使います。
- リモートプロジェクトとは、4D Server 上で開いた [.4DProject](#) ファイルのことです、リモートモードの 4D を使って接続します。4D Server は、プロジェクトの [圧縮形式](#) である .4dz ファイルをリモートの 4D に送信します。つまり、ストラクチャーファイルは読み取り専用です。この構成は通常、アプリケーションのテストに使います。



ただし、4D Server と同じマシンから接続している場合には、プロジェクトファイルの変更が可能です。この [特殊機能](#) により、クライアント/サーバーアプリケーションを運用時と同じコンテキストで開発することができます。

組み込みクライアント/サーバーアプリケーションを開く

ビルドされたクライアント/サーバーアプリケーションは起動や接続処理が簡易です：

- サーバーを起動するには、サーバーアプリケーションをダブルクリックします。プロジェクトファイルを選択する必要はありません。
- クライアントを起動するにも、同様にクライアントアプリケーションをダブルクリックします。すると、サーバーアプリケーションへの接続が直接おこなわれるため、

詳細については [アプリケーションビルド](#) ページを参照ください。

リモートプロジェクトを開く

4D Server 上で動いているプロジェクトに初めて接続する場合は、通常は標準の接続ダイアログを使います。以降は、最近使用したプロジェクトを開くメニュー や、4DLink ショートカットファイルを使って直接接続できるようになります。

4D Server で実行されているプロジェクトに接続するには：

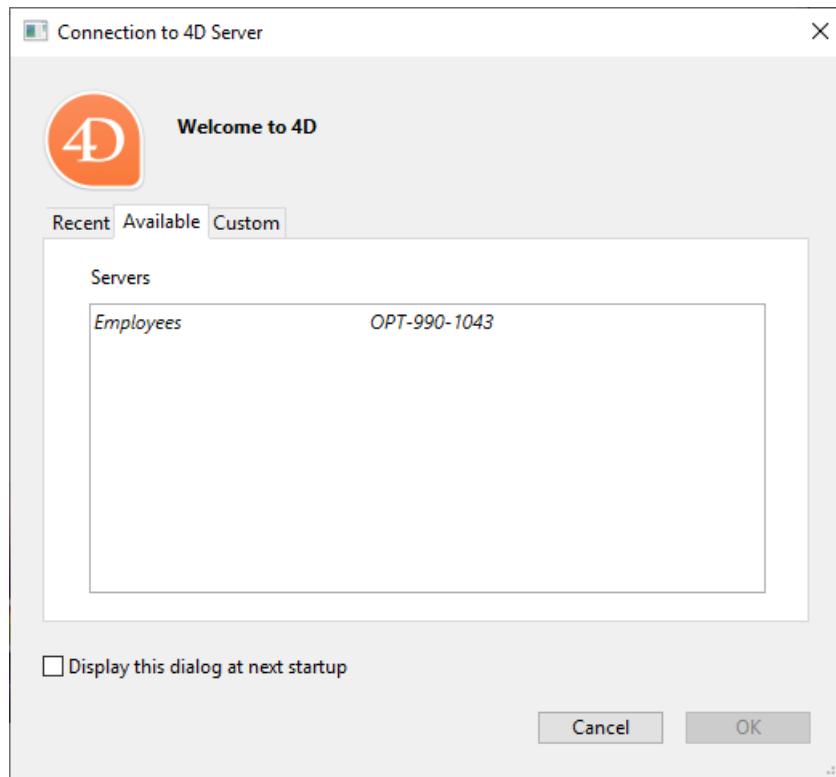
- Welcome ウィザードにて 4D Server に接続を選択します。

OR

ファイル メニューより 開く > リモートプロジェクト... を選択するか、開く ツールバー ボタンより同様に選択します。

4D Server に接続するためのダイアログが表示されます。ダイアログには 最近使用、利用可、および カスタム という、3つのタブがあります。

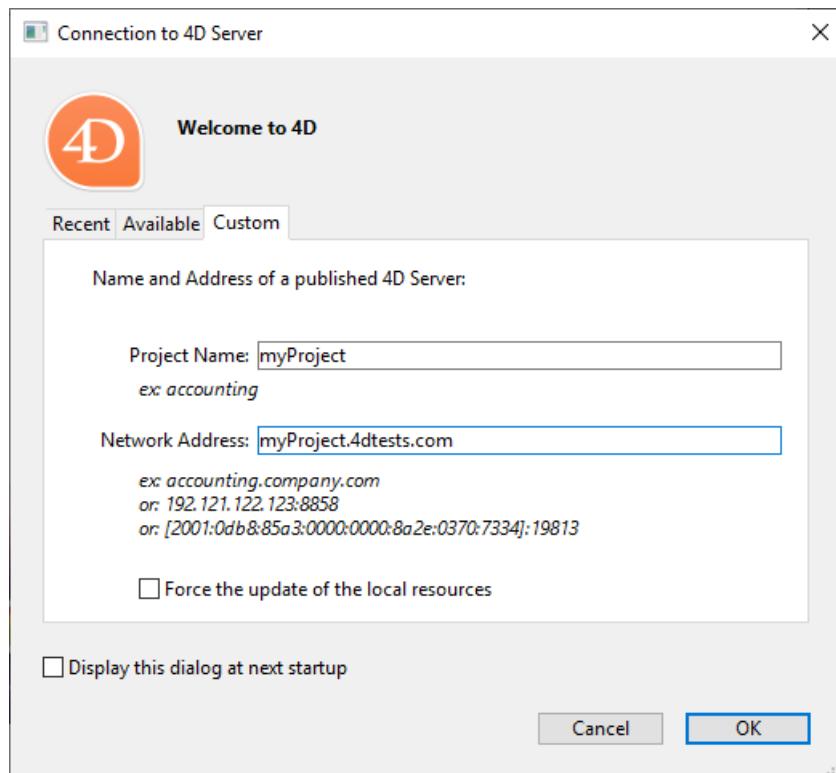
リモートの 4D と同じネットワークに 4D Server が接続されている場合は 利用可 タブを選択します。4D Server には組み込みの TCP/IP プロードキャストシステムがあり、デフォルトで、ネットワーク上に利用可能な 4D Server データベースの名前を公開します。このリストは、名前が見つかった順に表示され、動的に更新されます。



このリストからサーバーに接続するには、名前上でダブルクリックするか、名前を選択して OK ボタンをクリックします。

暗号化オプションが有効で公開されているデータベース名の前にはキャレット (^) が置かれます。

公開されているプロジェクトが 利用可 タブに見つからない場合には、カスタム タブを開きます。カスタムページでは、IPアドレスでネットワーク上のサーバーを指定し、それに任意の名前をつけられます。



- プロジェクト名: 4D Server プロジェクトのローカル名を指定できます。この名前は 最近使用 ページでプロジェクトを参照する際に使用されます。
- ネットワークアドレス: 4D Server が起動されたマシンの IPアドレスを指定します。

2つのサーバーが同じマシン上で同時に起動されているときは、IPアドレスの後にコロンとポート番号を続けます。例えば:
192.168.92.104:19820。

デフォルトで、4D Server の公開ポートは 19813 です。この番号は、プロジェクト設定で変更できます。

このページでサーバーを指定したら、OK ボタンをクリックしてサーバーに接続できます。

データベースが暗号化されて公開されている場合、名前の前にキャレット (^) を置かなければなりません。そうでなければ接続は拒否されます。
詳細はクライアント/サーバー接続の暗号化を参照してください。

サーバーとの接続が確立されると、そのリモートプロジェクトは 最近使用 タブのリストに加えられます。

サーバー上のプロジェクトファイルの更新

インターフリターモードの場合、4D Server は .4DProject プロジェクトファイル（非圧縮）の .4dz ファイルを自動的に作成し、リモートマシンに送信します。

- プロジェクトが編集され 4D Server にリロードされた場合など、必要に応じてプロジェクトの .4dz ファイルは自動的に更新されます。プロジェクトは次の場合一リロードされます：
 - 4D Server アプリケーションウィンドウが OS の最前面に来たり、同じマシン上の 4D アプリケーションが編集を保存した場合（後述参照）に自動でリロードされます。
 - RELOAD PROJECT コマンドが実行されたときにリロードされます。プロジェクトの新しいバージョンをソース管理システムよりプルしたときなどに、このコマンドを呼び出す必要があります。

リモートマシンのプロジェクトファイルの更新

4D Server 上で .4dz ファイルの更新版が生成された場合、その更新版を利用するには、接続中のリモート 4D マシンは一度ログアウトし、4D Server に再接続する必要があります。

4D と 4D Server の同じマシン上での使用

同じマシン上で 4D が 4D Server に接続すると、アプリケーションはシングルユーザー モードの 4D のようにふるまい、デザイン環境にてプロジェクトファイルの編集が可能です。この機能により、クライアント/サーバー アプリケーションを運用時と同じコンテキストで開発することができます。

デザイン環境にて 4D が すべてを保存 アクションを（ファイル メニューを使って明示的に、または、アプリケーションモードへの移行により暗示的に）おこなうと、4D Server は同期的にプロジェクトファイルをリロードします。4D Server によるプロジェクトファイルのリロードが完了するのを待って、4D は続行します。

ただし、[標準のプロジェクトアーキテクチャ](#) とは次のふるまいにおいて異なりますので、注意が必要です：

- 4D が使用する userPreferences.{username} フォルダーは、4D Server が使用するプロジェクトフォルダー内のものと同一ではありません。この専用の "userPreferences" フォルダーはプロジェクトシステムフォルダー内（つまり、.4dz プロジェクトを開く場合と同じ場所）に格納されます。
- 4D が使用する DerivedData フォルダーは、4D Server が使用するプロジェクトフォルダー内のものと同一ではありません。この専用の "DerivedDataRemote" フォルダーはプロジェクトのシステムフォルダー内に格納されます。
- catalog.4DCatalog ファイルは 4D ではなく 4D Server によって編集されます。catalog の情報はクライアント/サーバーリクエストによって同期されます。
- directory.json ファイルは 4D ではなく 4D Server によって編集されます。directory の情報はクライアント/サーバーリクエストによって同期されます。
- 4D は、4D Server 上のものではなく、独自の内部的なコンポーネントやプラグインを使用します。

プラグインやコンポーネントを 4D あるいは 4D Server アプリケーションレベルにインストールすることは、推奨されません。

コンポーネントの開発

4D のコンポーネントとは、[4D アプリケーションにインストール可能](#) な、1つ以上の機能を持つ 4D 関数やメソッド、フォームの一式です。たとえば、メールの送受信をおこない、それらを 4D アプリケーションに格納するための機能を持ったコンポーネントを作成できます。

ニーズに合わせて独自の 4D コンポーネントを開発し、それを非公開とすることができます。また、作成した [コンポーネントを4Dコミュニティで共有](#) することもできます。

定義

- マトリクスプロジェクト：コンポーネント開発に使用する 4D プロジェクト。マトリクスプロジェクトは特別な属性を持たない標準のプロジェクトです。マトリクスプロジェクトはひとつのコンポーネントを構成します。
- ホストプロジェクト：コンポーネントがインストールされ、それを使用するアプリケーションプロジェクト。
- コンポーネント：ホストアプリケーションによって使用される目的で、同アプリケーションの [Components](#) フォルダーにコピーされたマトリクスプロジェクト（コンパイル済みまたは [ビルド済み](#)）。

画面の説明

4D コンポーネントの作成とインストールは直接 4D を使用しておこないます：

- コンポーネントをインストールするには、[プロジェクトの Components フォルダー](#) にコンポーネントファイルをコピーします。エイリアスまたはショートカットも使用できます。
- 言い換れば、マトリクスプロジェクト自体も 1 つ以上のコンポーネントを使用できます。しかしコンポーネントが "サブコンポーネント" を使用することはできません。
- コンポーネントは次の 4D の要素を呼び出すことができます：クラス、関数、プロジェクトメソッド、プロジェクトフォーム、メニューバー、選択リストなど。反面、コンポーネントが呼び出せないものは、データベースメソッドとトリガーです。
- コンポーネント内でデータストアや標準のテーブル、データファイルを使用することはできません。しかし、外部データベースのメカニズムを使用すればテーブルやフィールドを作成し、そこにデータを格納したり読み出したりすることができます。外部データベースは、メインの 4D データベースとは独立して存在し、SQL コマンドでアクセスします。
- インターリューモードで動作するホストプロジェクトは、インターリューマまたはコンパイル済みどちらのコンポーネントも使用できます。コンパイルモードで実行されるホストデータベースでは、インターリューマのコンポーネントを使用できません。この場合、コンパイル済みコンポーネントのみが利用可能です。

ランゲージコマンドのスコープ

[使用できないコマンド](#) を除き、コンポーネントではすべての 4D ランゲージコマンドが使用できます。

コマンドがコンポーネントから呼ばれると、コマンドはコンポーネントのコンテキストで実行されます。ただし `EXECUTE METHOD` および `EXECUTE FORMULA` コマンドは除きます。また、ユーザー & グループテーマの読み出しコマンドはコンポーネントで使用することができますが、読み出されるのはホストプロジェクトのユーザー & グループ情報であることに注意してください（コンポーネントに固有のユーザー & グループはありません）。

`SET DATABASE PARAMETER` と `Get database parameter` コマンドは例外となります：これらのコマンドのスコープはグローバルです。これらのコマンドがコンポーネントから呼び出されると、結果はホストプロジェクトに適用されます。

さらに、`Structure file` と `Get 4D folder` コマンドは、コンポーネントで使用するための設定ができるようになっています。

`COMPONENT LIST` コマンドを使用して、ホストプロジェクトにロードされたコンポーネントのリストを取得できます。

使用できないコマンド

（読み取り専用モードで開かれるため）ストラクチャーファイルを更新する以下のコマンドは、コンポーネントで使用することができません。コンポーネント中で以下のコマンドを実行すると、-10511, "CommandName コマンドをコンポーネントでコールすることはできません" のエラーが生成されます：

- `ON EVENT CALL`
- `Method called on event`
- `SET PICTURE TO LIBRARY`

- REMOVE PICTURE FROM LIBRARY
- SAVE LIST
- ARRAY TO LIST
- EDIT FORM
- CREATE USER FORM
- DELETE USER FORM
- CHANGE PASSWORD
- EDIT ACCESS
- Set group properties
- Set user properties
- DELETE USER
- CHANGE LICENSES
- BLOB TO USERS
- SET PLUGIN ACCESS

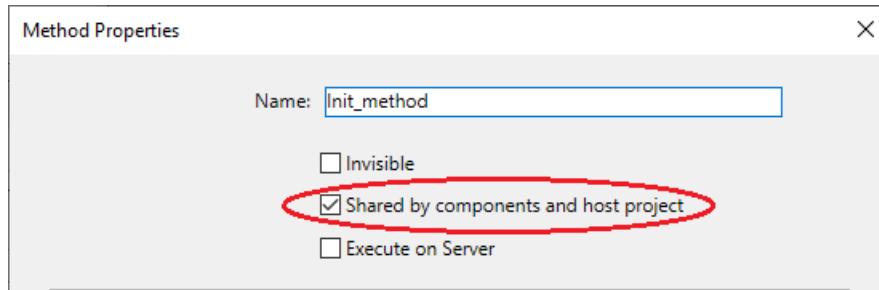
注:

- `Current form table` コマンドは、プロジェクトフォームのコンテキストで呼び出されると `Nil` を返します。ゆえにこのコマンドをコンポーネントで使用することはできません。
- SQLデータ定義言語のコマンド（`CREATE TABLE`、`DROP TABLE` 等）をコンポーネントのフレームワークで使用することはできません。ただし、外部データベースの場合は使用することができます（`CREATE DATABASE` SQL コマンド参照）。

プロジェクトメソッドの共有

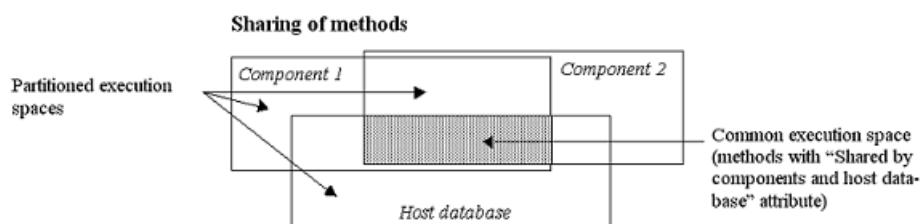
マトリクスプロジェクトのすべてのプロジェクトメソッドは、コンポーネントに含まれます。つまり、マトリクスプロジェクトをコンポーネント化した後、これらのプロジェクトメソッドは同コンポーネント内で呼び出して実行することができます。

他方、デフォルトでは、これらのプロジェクトメソッドはホストプロジェクトに表示されず、呼び出すこともできません。マトリクスプロジェクトで、メソッドプロパティダイアログボックスのコンポーネントとホストプロジェクト間で共有 ボックスをチェックすることで、ホストプロジェクトと共有したいメソッドを明示的に設定することができます。



設定することで、共有されたプロジェクトメソッドはホストプロジェクトにて呼び出すことができるようになります（しかしホストプロジェクトのメソッドエディターで編集することはできません）。これらのメソッドはコンポーネントの エントリーポイントとなります。

セキュリティのため、デフォルトでは、コンポーネントはホストプロジェクトのプロジェクトメソッドを実行することはできません。特定の場合に、ホストプロジェクトのプロジェクトメソッドにコンポーネントがアクセスできるようにする必要があるかもしれません。そうするには、ホストプロジェクトのプロジェクトメソッド側で、コンポーネントからのアクセスを可能にするよう明示的に指定しなければなりません。これはメソッドプロパティダイアログボックスの、コンポーネントとホストプロジェクト間で共有 で設定します。



ホストプロジェクトのプロジェクトメソッドがコンポーネントから利用可能になっていれば、`EXECUTE FORMULA` または `EXECUTE METHOD` コマンドを使用して、コンポーネント側からホストのメソッドを実行することができます。たとえば：

```
// ホストメソッド  
component_method("host_method_name")
```

```
// コンポーネントメソッド  
C_TEXT($1)  
EXECUTE METHOD($1)
```

インタープリターコンポーネントがインストールされたインターパリターホストデータベースは、それがインターパリターコンポーネントのメソッドを呼び出さなければ、コンパイル/シンタックスチェックができます。そうでない場合、コンパイルまたはシンタックスチェックを実行しようとすると警告ダイアログが表示され、操作を実行することはできません。

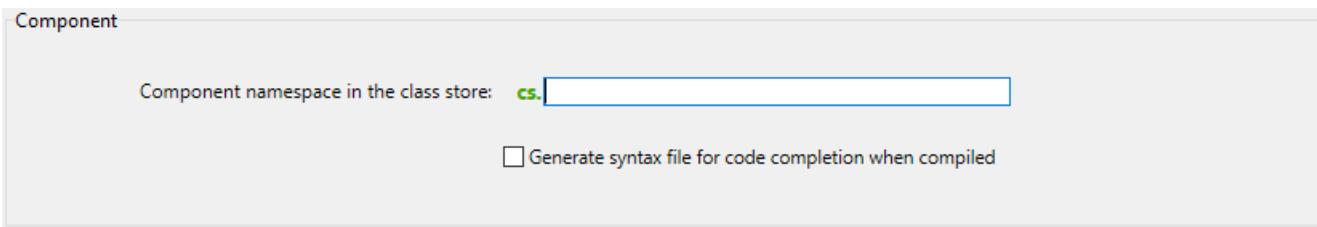
一般的に、インターパリターメソッドはコンパイル済みメソッドを呼び出せますが、逆はできません。これをおこなうには `EXECUTE METHOD` や `EXECUTE FORMULA` コマンドを使用します。

クラスや関数の共有

デフォルトでは、ホストプロジェクトの 4Dメソッドエディターからコンポーネントのクラスと関数を呼び出すことはできません。コンポーネントのクラスと関数をホストプロジェクトに公開したい場合は、コンポーネント名前空間を宣言する必要があります。また、コンポーネントのクラスや関数がホストメソッドエディターでどのように提案されるかをコントロールすることもできます。

コンポーネント名前空間の宣言

コンポーネントのクラスや関数をホストプロジェクトに公開するには、マトリクスプロジェクトの設定の [一般ページにある「クラスストア内でのコンポーネント名前空間 オプション](#) に値を入力します。デフォルトでは、このエリアは空です。つまり、コンポーネントのクラスはコンポーネント外で利用できません。



名前空間 は、同じ名前のクラスや関数を持つ異なるコンポーネントがホストプロジェクトで使用されている場合に、競合が発生しないようにします。コンポーネント名前空間は、[プロパティの命名規則](#) に準拠する必要があります。

値を入力すると、ホストプロジェクトのコードにおいてユーザークラスストア (cs) 内の cs.<値> 名前空間を介して、コンポーネントのクラスと関数が利用可能になることを宣言することになります。たとえば、`getArea()` 関数を持つ `Rectangle` クラスが存在する場合に、コンポーネント名前空間として "eGeometry" を入力すると、このプロジェクトがコンポーネントとしてインストールされると、ホストプロジェクトの開発者は次のように記述することができます：

```
// ホストプロジェクトにて  
var $rect: cs.eGeometry.Rectangle  
$rect:=cs.eGeometry.Rectangle.new(10;20)  
$area:=$rect.getArea()
```

競合を避けるためには、優れた識別名の使用が推奨されます。もし、コンポーネントと同じ名前のユーザークラスがすでにプロジェクトに存在していた場合、そのユーザークラスが考慮され、コンポーネントクラスは無視されます。

コンポーネントの ORDAクラスは、ホストプロジェクトでは使用できません。たとえば、コンポーネントに `Employees` というデータクラスがある場合、ホストプロジェクトで "cs.Mycomponent.Employee" クラスを使用することはできません。

非表示クラス

アンダースコア ("_") を名前の前に付けることで、コンポーネントのクラスや関数を非表示にすることができます。コンポーネント名前空間が定義されている場合、非表示のクラスや関数はコード補完の際に提案されません。

ただし、名前がわかっていていれば使用することは可能です。たとえば、`_Rectangle` クラスが非表示の場合でも、次のシンタックスは有効です：

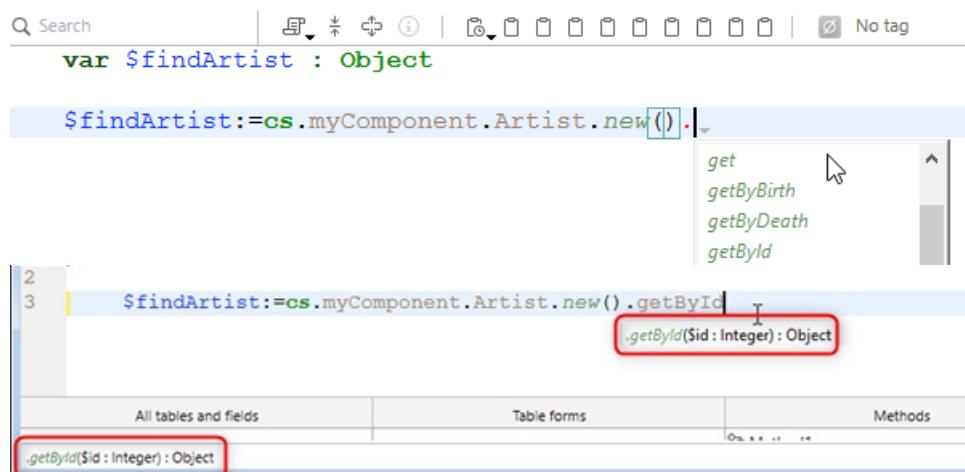
```
$rect:=cs.eGeometry._Rectangle.new(10;20)
```

非表示のクラス内の、非表示でない関数は、そのクラスを [継承](#) するクラスでコード補完を使用すると提案されます。たとえば、あるコンポーネントに `_Person` クラスを継承した `Teacher` クラスがある場合、`Teacher` のコード補完では `_Person` の非表示でない関数が提案されます。

コンパイル済みコンポーネントのコード補完

開発者に向けてコンポーネントを使いやすくするため、マトリクスプロジェクトの設定で、一般ページの [コンパイル時にコード補完用のシンタックスファイルを生成するオプション](#) をチェックすることができます。

すると、コンパイル時にシンタックスファイル (JSON形式) が自動生成され、そこにコンポーネントのクラス、関数、および [公開メソッド](#) のシンタックスを格納し、コンポーネントプロジェクトの `¥Resources¥ja.lproj` フォルダーに配置されます。4D は、このシンタックスファイルをもとに、コード補完や関数シンタックスなどのコードエディター用のヘルプを生成します。



[コンポーネント名前空間](#) を入力しない場合、シンタックスファイル生成のオプションがチェックされていても、クラスや公開メソッド用のリソースは生成されません。

変数の渡し方

ローカル、プロセス、インタープロセス変数は、コンポーネントとホストプロジェクト間で共有されません。ホストプロジェクトからコンポーネントの変数を編集、またはその逆をおこなう唯一の方法はポインターを使用することです。

配列を使用した例：

```
// ホストプロジェクト側:  
ARRAY INTEGER(MyArray;10)  
AMethod(>MyArray)  
  
// コンポーネント側で AMethod プロジェクトメソッドは以下の通りです:  
APPEND TO ARRAY($1->;2)
```

変数を使用した例：

```
C_TEXT(myvariable)  
component_method1(>myvariable)
```

```
C_POINTER($p)
$p:=component_method2(...)
```

ポインターを使用しない場合でも、コンポーネント側からホストデータベースの（変数そのものではなく）変数の値にアクセスすること自体は可能ですし、その逆も可能です：

```
// ホストデータベース内
C_TEXT($input_t)
$input_t:="DoSomething"
component_method($input_t)
// component_method は $1 に "DoSomething" を受け取ります ($input_t 変数を受け取るわけではありません)
```

ホストプロジェクトとコンポーネント間でポインターを使用して通信をおこなうには、以下の点を考慮する必要があります：

- Get pointer をコンポーネント内で使用した場合、このコマンドはホストプロジェクトの変数へのポインターを返しません。また逆にこのコマンドをホストプロジェクトで使用した場合も同様です。
- コンパイル済みプロジェクトでは、コンパイルされたコンポーネントしか使用できませんが、インタープリタープロジェクトの場合には、インターパリタおよびコンパイル済みコンポーネントを同時に使用することができます。この場合、ポインターの利用は以下の原則を守らなければなりません：インターパリターモードでは、コンパイルモードにおいて作成されたポインターを解釈できます。逆にコンパイルモードでは、インターパリターモードにて作成されたポインターを解釈することはできません。以下の例でこの原則を説明します：同じホストプロジェクトにインストールされた 2つのコンポーネント C (コンパイル済) と I (インターパリタ) があります：
- コンポーネントC が定義する変数 myCvar があるとき、コンポーネントI はポインター ->myCvar を使用して変数の値にアクセスすることができます。
- コンポーネントI が定義する変数 myIvar があるとき、コンポーネントC はポインター ->myIvar を使用しても変数の値にアクセスすることはできません。このシンタックスは実行エラーを起こします。
- RESOLVE POINTER を使用したポインターの比較はお勧めできません。変数の分離の原則により、ホストプロジェクトとコンポーネント（あるいは他のコンポーネント）で同じ名前の変数が存在することができますが、根本的にそれらは異なる内容を持ちます。両コンテキストで、変数のタイプが違うことさえあります。ポインター myptr1 と myptr2 がそれぞれ変数を指すとき、以下の比較は正しくない結果となるかもしれません：

```
RESOLVE POINTER(myptr1;vVarName1;vtablenum1;vfieldnum1)
RESOLVE POINTER(myptr2;vVarName2;vtablenum2;vfieldnum2)
If(vVarName1=vVarName2)
// 変数が異なっているにもかかわらず、このテストはtrue を返します
```

このような場合には、ポインターを比較しなければなりません：

```
If(myptr1==myptr2) // このテストはFalse を返します
```

エラー処理

ON ERR CALL コマンドによって実装された エラー処理メソッド は、実行中のプロジェクトに対してのみ適用されます。コンポーネントによって生成されたエラーの場合、ホストプロジェクトの ON ERR CALL エラー処理メソッドは呼び出されず、その逆もまた然りです。

ホストプロジェクトのテーブルへのアクセス

コンポーネントでテーブルを使用することはできませんが、ホストプロジェクトとコンポーネントはポインターを使用して通信をおこなうことができます。たとえば、以下はコンポーネントで実行可能なメソッドです：

```
// コンポーネントメソッドの呼び出し
methCreateRec(->[PEOPLE];->[PEOPLE]Name;"Julie Andrews")
```

コンポーネント内の `methCreateRec` メソッドのコード:

```
C_POINTER($1) // ホストプロジェクトのテーブルへのポインター  
C_POINTER($2) // ホストプロジェクトのフィールドへのポインター  
C_TEXT($3) // 代入する値  
  
$tablepointer:=$1  
$fieldpointer:=$2  
CREATE RECORD($tablepointer->)  
  
$fieldpointer->:=$3  
SAVE RECORD($tablepointer->)
```

コンポーネントのコンテキストにおいて、テーブルフォームへの参照はすべてホスト側のテーブルフォームへの参照だと 4D はみなします（コンポーネントはテーブルを持つことができないからです）。

テーブルやフィールドの利用

コンポーネントは、マトリクスプロジェクトのストラクチャーで定義されたテーブルやフィールドを使用することはできません。しかし外部データベースを作成し、そのテーブルやフィールドを必要に応じ利用することはできます。外部データベースの作成と管理は SQL を用いておこないます。外部データベースは、メインの4Dプロジェクトから独立している別の4Dプロジェクトですが、メインプロジェクトから操作が可能です。外部データベースの利用は、そのデータベースを一時的にカレントデータベースに指定することです。言い換えれば、4Dが実行するSQLクエリのターゲットデータベースとして外部データベースを指定します。外部データベースの作成はSQLの `CREATE DATABASE` コマンドを使用します。

例題

以下のコードはコンポーネントに実装されており、外部データベースに対して3つの基本的なアクションをおこないます：

- 外部データベースを作成します（存在しない場合）
- 外部データベースにデータを追加します
- 外部データベースからデータを読み込みます

外部データベースの作成：

```
<>MyDatabase:=Get 4D folder+"\MyDB" // (Windows) データを許可されているディレクトリに保存します  
Begin SQL  
    CREATE DATABASE IF NOT EXISTS DATAFILE :[<>MyDatabase];  
    USE DATABASE DATAFILE :[<>MyDatabase];  
    CREATE TABLE IF NOT EXISTS KEEPIT  
    (  
        ID INT32 PRIMARY KEY,  
        kind VARCHAR,  
        name VARCHAR,  
        code TEXT,  
        sort_order INT32  
    );  
  
    CREATE UNIQUE INDEX id_index ON KEEPIT (ID);  
  
    USE DATABASE SQL_INTERNAL;  
  
End SQL
```

外部データベースへのデータ書き込み：

```

$Ptr_1:=$2 // ポインターを介してホストプロジェクトからデータを取得します
$Ptr_2:=$3
$Ptr_3:=$4
$Ptr_4:=$5
$Ptr_5:=$6
Begin SQL

    USE DATABASE DATAFILE :[<>MyDatabase];

    INSERT INTO KEEPIT
    (ID, kind, name, code, sort_order)
    VALUES
    (:[$Ptr_1], :[$Ptr_2], :[$Ptr_3], :[$Ptr_4], :[$Ptr_5]);

    USE DATABASE SQL_INTERNAL;

End SQL

```

外部データベースからデータを読み込み:

```

$Ptr_1:=$2 // ホストプロジェクトへのデータアクセスはポインターを通じておこないます
$Ptr_2:=$3
$Ptr_3:=$4
$Ptr_4:=$5
$Ptr_5:=$6

Begin SQL

    USE DATABASE DATAFILE :[<>MyDatabase];

    SELECT ALL ID, kind, name, code, sort_order
    FROM KEEPIT
    INTO :$Ptr_1, :$Ptr_2, :$Ptr_3, :$Ptr_4, :$Ptr_5;

    USE DATABASE SQL_INTERNAL;

End SQL

```

フォームの使用

- 特定のテーブルに属さない"プロジェクトフォーム"のみが、コンポーネント内で利用できます。マトリクスプロジェクトのすべてのプロジェクトフォームをコンポーネントで使用することができます。
- コンポーネントはホストプロジェクトのテーブルフォームを使用できます。この場合、コンポーネントのコードでフォームを指定するにあたっては、テーブル名ではなく、テーブルへのポインターを使用しなければならないことに注意してください。

コンポーネントが `ADD RECORD` コマンドを使用すると、ホストプロジェクトのコンテキストで、ホストプロジェクトのカレントの入力フォームが表示されます。したがって、その入力フォーム上に変数が含まれている場合、コンポーネントはその変数にアクセスできません。

- コンポーネントフォームをホストプロジェクト内でサブフォームとして公開することができます。これは具体的には、グラフィックオブジェクトを提供するコンポーネントを開発できることを意味します。たとえば、4D社が提供するウィジェットはコンポーネントのサブフォーム利用に基づいています。

コンポーネントのコンテキストにおいては、参照されるプロジェクトフォームはすべてコンポーネント内に存在している必要があります。たとえば、コンポーネント内において、`DIALOG` または `Open form window` コマンドを使用してホスト側のプロジェクトフォームを参照しようとした場合にはエラーが生成されます。

リソースの使用

コンポーネントは、自身の Resources フォルダーにあるリソースを使用することができます。

これによって自動メカニズムが有効となり、コンポーネントの Resources フォルダー内で見つかった XLIFF ファイルは、同コンポーネントによってロードされます。

1つ以上のコンポーネントを含むホストプロジェクトでは、ホストプロジェクトと同様にそれぞれのコンポーネントも固有のリソースチェーンを持っています。リソースは異なるプロジェクト間で分離されます。コンポーネント A のリソースにコンポーネント B やホストプロジェクトからアクセスすることはできません。

初期化のコードの実行

コンポーネントは、ホストデータベースを開いたときまたは閉じたときに、自動的に 4D コードを実行することができます。これによってたとえば、ホストデータベースに関連する設定やユーザーの状態などを読み込み・保存することができます。

初期化やデータベースを閉じるコードの実行は、`On Host Database Event` データベースメソッドを使用しておこなわれます。

セキュリティ上の理由から、`On Host Database Event` データベースメソッドを使用可能にするためには、その実行をホストデータベースで明示的に許可する必要があります。セキュリティ上の理由から、`On Host Database Event` データベースメソッドを使用可能にするためには、その実行をホストデータベースで明示的に許可する必要があります。

コンポーネントの保護: コンパイル

コンポーネントとしてインストールされたマトリクスプロジェクトのコードは、ホストプロジェクトからデフォルトでアクセス可能です。特に:

- エクスプローラーのメソッドページに存在する共有のプロジェクトメソッドは、ホストプロジェクトのメソッドから呼び出し可能です。エクスプローラーのプレビューエリアでそれらの内容を選択してコピーすることができます。また、その内容はデバッガーで見ることもできます。しかし、それらをメソッドエディター上で開いたり編集したりすることはできません。
- マトリクスプロジェクトの他のプロジェクトメソッドはエクスプローラーに現れません。しかし、ホストプロジェクトのデバッガーには内容が表示されます。
- 名前空間が宣言されいれば、非表示でないクラスや関数はデバッガーで確認することができます。**

コンポーネントのコードを効果的に保護するには、マトリクスプロジェクトを [コンパイルしビルドして](#)、.4dz ファイルとして提供します。コンパイルされたマトリクスプロジェクトがコンポーネントとしてインストールされると:

- 共有のプロジェクトメソッド、クラス、および関数は、ホストプロジェクトのメソッドから呼び出し可能です。共有のプロジェクトメソッドは、エクスプローラーのメソッドページにも表示されます。しかし、その内容はプレビューエリアにもデバッガーにも表示されません。
- マトリクスプロジェクトの他のプロジェクトメソッドは一切表示されません。

コンポーネントの共有

開発したコンポーネントを [GitHub](#) で公開し、4D 開発者のコミュニティをサポートすることをお勧めします。正しく参照されるためには、`4d-component` トピックをご利用ください。

プラグインの開発

プラグインの必要性

オブジェクトやレコードの操作、ユーザーインターフェースの実装のため、4D は数百のビルトインコマンドを提供していますが、さらに特殊な機能（プラットフォーム依存のものなど）が必要になることがあります。たとえば、Windows上のODBC、macOS上のアップルサービス、特殊な統計機能、ソーシャルネットワークログイン、決済用のプラットフォーム、ネットワークを介したファイルアクセス、特殊なユーザーインターフェース、独自のピクチャー構造などです。

これらの機能をすべて、macOS と Windows の両方で 4D コマンドによって提供しようとした場合、コマンド数は数千に膨れ上がると同時に、ほとんどのユーザーはそれらの追加機能を必要としないでしょう。また、そのような万能ツールを作り上げたとしても、その結果として 4D 環境は複雑化することになり、4D の学習が困難になるだけでなく、成果が得られるまで時間をするようになるでしょう。

4D 環境のモジュール性により、基礎的なアプリケーションの作成はもちろんのこと、非常に複雑なシステムの開発も可能です。4D プラグインのアーキテクチャによって、4D 環境はあらゆるアプリケーションとユーザーに対して開かれています。4D プラグインによってアプリケーションやユーザーの生産性を飛躍させることができます。

プラグインとは何か

プラグインとは、4D 起動時にロードされるコードのことです。プラグインは、4D に機能を追加します。

通常、プラグインは：

- 4D ができないことを処理します（プラットフォーム特有の技術など）
- 4D だけでは難しいことを実現します
- プラグインのエントリーポイントの形でのみ提供されている機能を提供します

プラグインには通常複数のルーチンが含まれています。プラグインは外部エリアを操作でき、外部プロセスを実行できます。

- プラグインルーチンとは、ネイティブ言語（通常は C あるいは C++）で書かれたルーチンで、なんらかの処理を実行します。
- 外部エリアとはフォームの一部で、あらゆるもの表示することができ、必要に応じてユーザー操作を受け付けることができます。
- 外部プロセスとは、通常はループ形式で単独実行されるプロセスのことです。プロセスのコードはすべてプラグインに属しており、4D はプロセスに対してイベントを送受信するだけです。

重要な注記

プラグインは、小さな処理をおこなう一つのルーチンを実行するだけの、とても簡単なものであります。また、百以上のルーチンとエリアを扱うような、非常に複雑なものもあります。プラグインの機能に制限はありませんが、プラグインの開発にあたっては、プラグインがあくまで従たるコードであることに留意が必要です。プラグインは 4D 内で実行されます。プラグインは独立したアプリケーションではなく、4D の一部です。プラグインは、他のプラグインや 4D 自身と CPU 時間とメモリを共有します。したがって、プラグインのコードは、必要なリソースだけを使用する控えめなコードであるべきです。たとえば、非常に長いループ処理においては（その重要性が絶対的な優先権を要求しないかぎり）、プラグインは `PA_Yield()` を呼び出して、4D のスケジューラーにも処理時間を割り当てるべきです。

プラグインの作り方

4D は GitHub 上に、4D Plugin API と the 4D Plugin Wizard を含んだオープンソースの [プラグイン SDK](#) (英語) を提供しています：

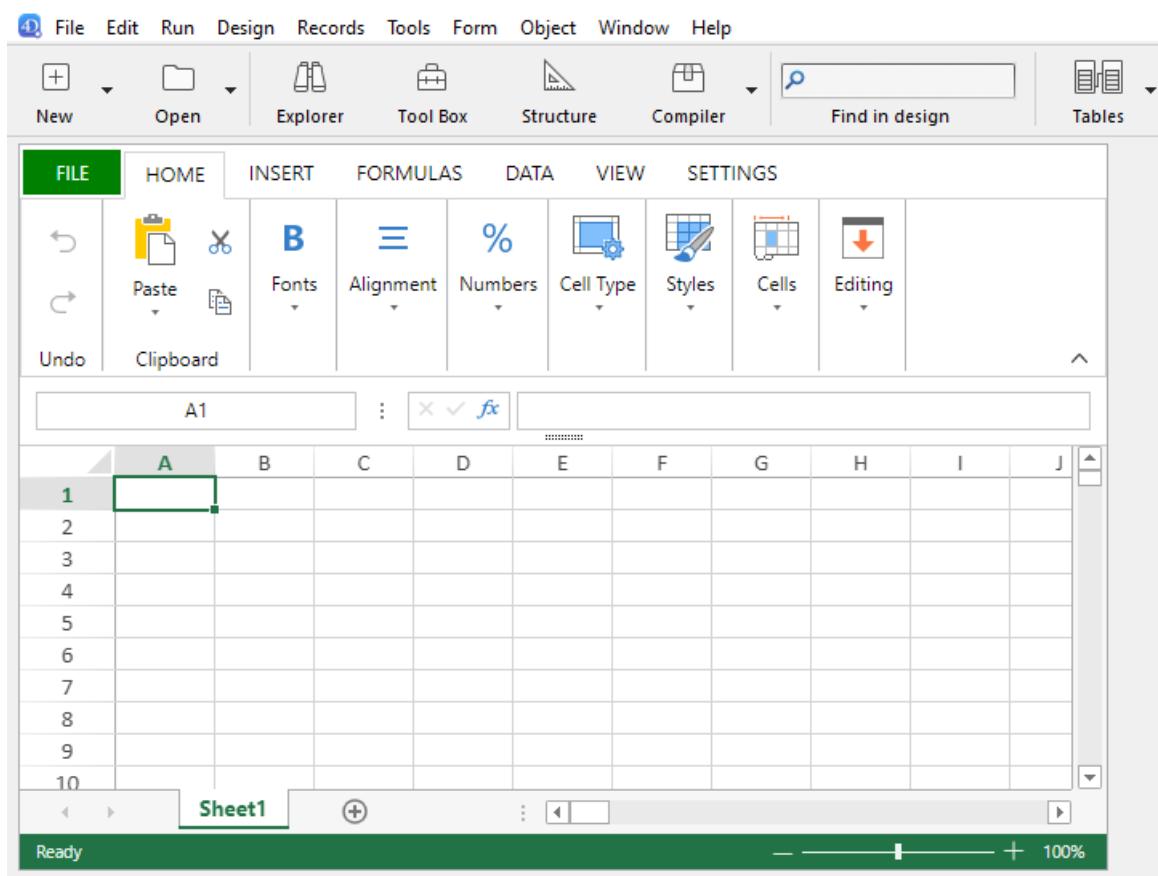
- [4D Plugin API](#) は C で書かれており、プラグインの開発を助けるための機能を400以上追加します。4D Plug-in API の機能は、4D とプラグイン間の通信を管理します。
- [4D Plugin Wizard](#) は、4D プラグインの開発を簡略化するために不可欠なツールです。プラグインのロードや、プラグインとの通信に 4D が必要とするコードがツールによって提供されることで、デベロッパーはプラグインの根幹をなすコードに集中することができます。

プラグインの共有

開発したプラグインを [GitHub](#) で公開し、4D開発者のコミュニティをサポートすることをお勧めします。正しく参照されるためには、[4d-plugin](#) トピックをご利用ください。

はじめに

4D View Pro は、[4D フォームエリア](#)と専用の [メソッド](#)が含まれる [4Dコンポーネント](#)です。これにより、先進的なスプレッドシート機能をプロジェクトに埋め込むことが可能です。



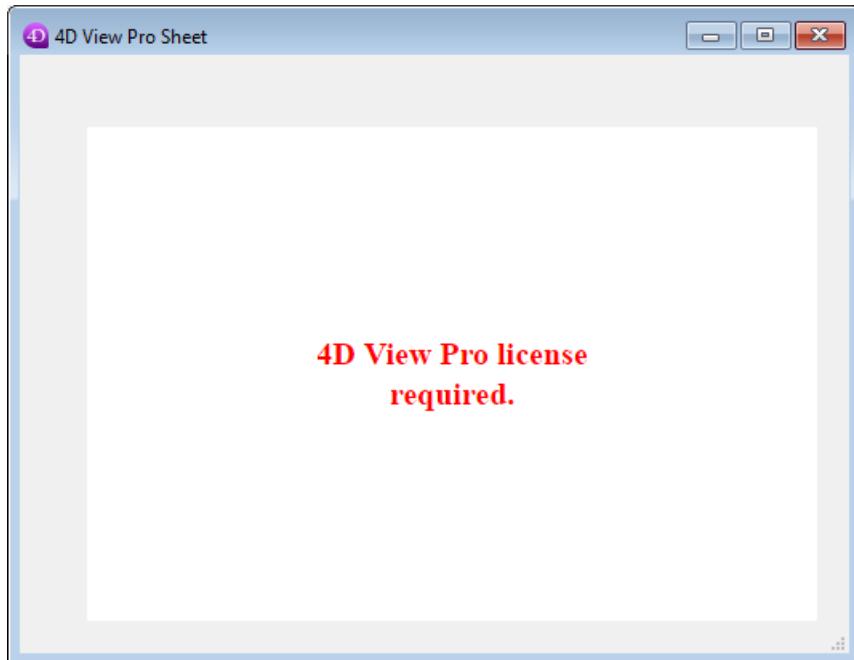
スプレッドシートとは、セルのグリッドを格納したアプリケーションのことです、これらのセルに情報を入力したり、計算を実行させたり、あるいはピクチャーを表示したりすることができます。4D View Proは、4Dに統合された [SpreadJS スプレッドシートソリューション](#)に基づいて動作します。

フォームに 4D View Pro エリアを埋め込むことで、4D View Pro コマンドを使ってスプレッドシートドキュメントを読み込んだり書き出したりすることができます。

インストールとアクティベーション

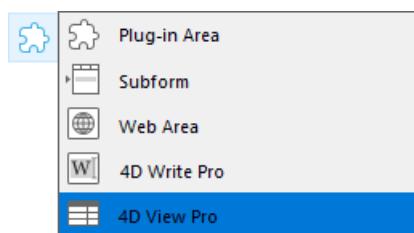
4D View Pro 機能は 4D に直接含まれているため、配布と運用が容易です。追加のインストールは必要ありません。

しかしながら、4D View Pro はライセンスを必要とします。これらの機能を使用するには、アプリケーションにおいて当該ライセンスを有効化しておく必要があります。4D View ライセンスがインストールされていない場合、4D View Pro 機能を必要とするオブジェクトのコンテンツはランタイムでは表示されず、エラーメッセージだけが表示されます：



4D View Pro エリアを挿入する

4D View Pro ドキュメントは 4D View Pro という名前の [4Dフォームオブジェクト](#) 内に表示され、手動で編集されます。このオブジェクトを選択するには、オブジェクトバーの最後のツールをクリックします：



また、[オブジェクトライブラリ](#) であらかじめ設定された 4D View Pro エリアを選択することもできます。

4D View Pro エリアは、[オフスクリーンでも作成・使用することができます。](#)

[エリアの設定](#) は、プロパティリストと 4D View Pro メソッドを使っておこないます。

セレクション、入力、およびナビゲーションの基本

スプレッドシートは行と列から構成されています。各行には番号が割り当てられています。各列には文字（アルファベットの文字数を超えた場合には文字のグループ）が割り当てられています。行と列の交差する場所がセルと呼ばれます。セルを選択したり、そのコンテンツを編集したりすることができます。

セル、列、および行の選択

- セルを選択するには、単にセルをクリックするか、キーボードの矢印キーを使用します。その中身（あるいは式）がセル内に表示されます。
- 複数の連続したセルを選択するには、マウスをセレクションの端から端へとドラッグします。また、Shiftキーを押しながらセレクションの二つの端をクリックすることでも選択可能です。
- スプレッドシート内のセルをすべて選択するには、エリアの左上端にあるセルをクリックします：



- 列を選択するには、対応する文字（アルファベット）をクリックします。
- 行を選択するには、対応する番号をクリックします。
- 連続していないセルを複数選択するには、Ctrlキー（Windows）あるいはCommandキー（macOS）を押しながら、選択したいセルをそれぞれクリックします。

- セルの選択を解除するには、スプレッドシート内のどこかをクリックすれば選択解除されます。

データの入力

セルをダブルクリックすると、そのセル内で入力モードに入ります。セルが空でない場合、挿入カーソルはセルのコンテンツの最後に置かれます。



セルが選択されていれば、たとえ挿入カーソルが非表示であってもデータを直接入力することができます。その場合、入力した内容はセルのコンテンツを上書きします。

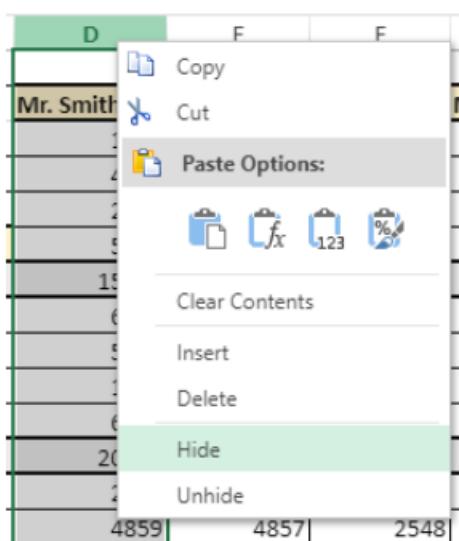
Tabキーはセルの内容を確定し、一つ右のセルを選択します。Shift + Tabキーの組み合わせでは、セル入力を確定したあと、一つ左のセルを選択します。

キャリッジリターンキーはセルの入力を確定し、一つ下のセルを選択します。Shift + キャリッジリターンキーの組み合わせで、セル入力を確定したあと、一つ上のセルを選択します。

方向キー (矢印) を使用すると、矢印の方向へとセルの選択を移動することができます。

コンテキストメニューの使い方

4D View Pro エリアでは、コピー/ペーストといった標準の編集機能だけでなく、基本的なスプレッドシート機能も備えている自動コンテキストメニューを利⽤することができます:



コンテキストメニューのコピー/カット/ペースト機能はスプレッドシートエリア内でのみ動作し、システムのペーストボードにはアクセスしません。しかしながら、Ctrl+c/Ctrl+v といったシステムショートカットは動作し、エリアと他のアプリケーション間でデータを交換するために使用することができます。

クリックしたエリアに応じて、メニューには次の選択肢が表示されます:

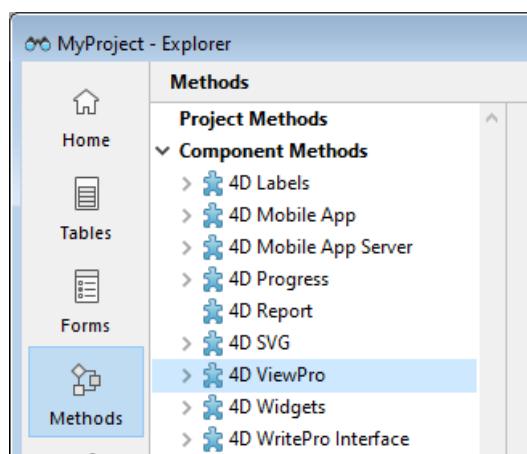
- 列や行のヘッダーをクリックした場合: コンテンツの挿入、削除、非表示、再表示
- セルあるいはセルレンジのクリック:
 - フィルタリング: フィルタリングを使用して行を非表示にします (SpreadJS ドキュメントの [Filtering rows](#) を参照ください)
 - ソート: 列のコンテンツを並べ替えます。
 - コメントの挿入: ユーザーコメントを入力できます。コメントが入力されているセルには、小さな赤い三角形が表示されます:

8355	7195
------	------

4D View Pro メソッドの使い方

4D View Pro メソッドは、ほかの 4D ランゲージコマンドと同様に 4D のメソッドエディターにて使用することができます。

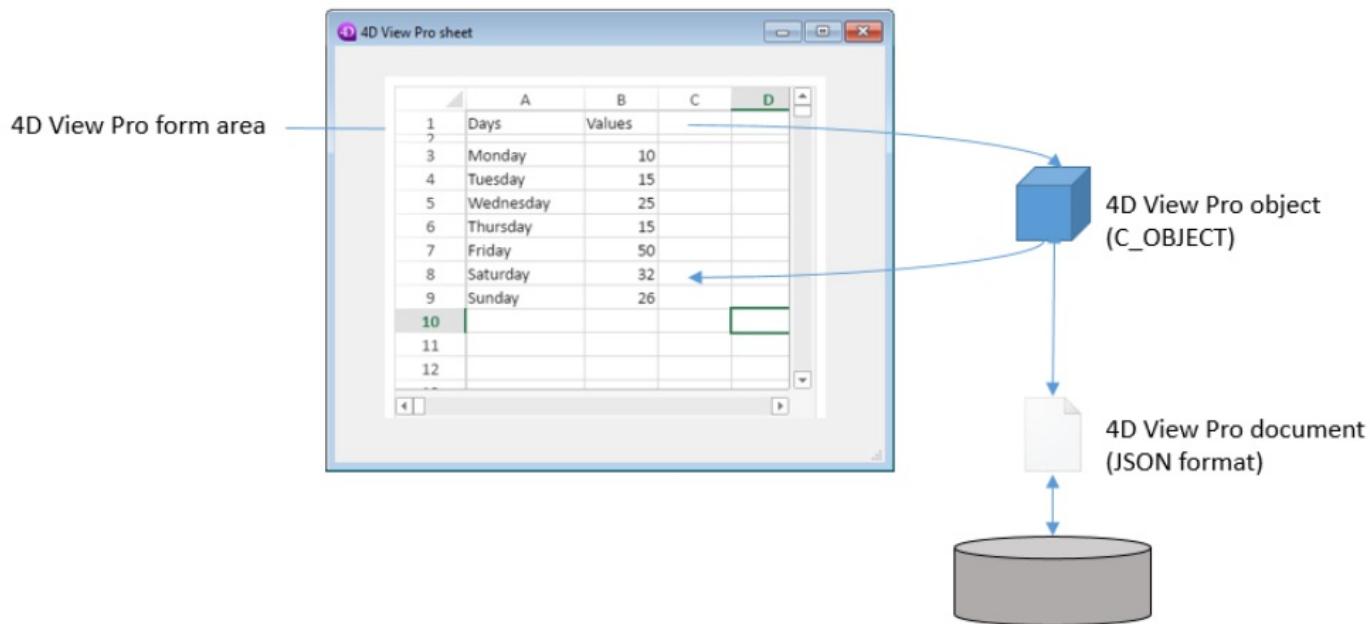
4D View Pro はビルトインの 4Dコンポーネントであるため、そのメソッドの一覧はエクスプローラーにおいて、メソッドページの コンポーネントメソッド 内に表示されます：



コンポーネントメソッドの詳細な一覧については [メソッドリスト](#) を参照ください。

4D View Proのエリアの操作

4D View Pro エリアは、複数のオブジェクトや要素を扱います。



ほとんどの 4D View Pro メソッドは、[4D View Pro のフォームエリア名](#) (4Dフォームオブジェクト) を `vpAreaName` 引数として必要とします。このエリア名は、[オブジェクト名](#) プロパティの値のことです。

たとえば、"myVpArea" という名前の 4D View Pro エリアの列数を設定するには、次のように書きます：

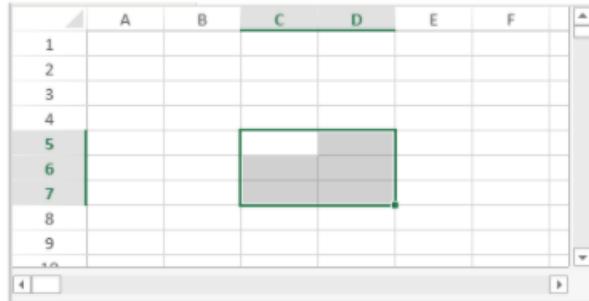
```
VP SET COLUMN COUNT("myVpArea";5)
```

フォームエリアへの 4D View Pro オブジェクトの読み込みが完了すると、4D は [On VP Ready](#) フォームイベントを生成します。エリアを操作する 4D View Pro コードはこのイベント内で実行する必要があります。そうでない場合はエラーが返されます。

レンジオブジェクトの使い方

4D View Pro のメソッドの中には、`rangeObj` 引数を必要とするものがあります。4D View Pro におけるレンジとは、スプレッドシート内の特定のエリアを参照するオブジェクトです。このエリアは、1つ以上のセルで構成されます。4D View Pro メソッドを使って、作成したレンジを他のメソッドに受け渡し、ドキュメントの特定箇所に対して読み取りや書き込み操作をおこなうことができます。

たとえば、次のセルに対応するレンジオブジェクトを作成したい場合：



[VP Cells](#) メソッドを使用できます:

```
var $myRange : Object
$myRange:=VP Cells("ViewProArea";2;4;2;3) // C5 から D7
```

その後、\$myRange を別の 4D View Pro メソッドに渡して、これらのセルを変更することができます (たとえば、[VP SET BORDER](#) で境界線を追加します)。

4D View Pro のレンジオブジェクトは、いくつかのプロパティで構成されます:

- area - 4D View Pro エリアの名称
- ranges - レンジオブジェクトのコレクション。各レンジオブジェクトで利用できるプロパティは、レンジオブジェクトの種類によって異なります。たとえば、列のレンジオブジェクトには .column と .sheet プロパティしか含まれません。

プロパティ		タイプ	説明	利用可能な対象
area		テキスト	4D View Pro フォームオブジェクト名	常に利用可能
ranges		collection	レンジのコレクション	常に利用可能
[].name		テキスト	レンジ名	name
[].sheet		number	シートのインデックス (デフォルトではカレントシートのインデックス; 0起点)	cell, cells, row, rows, column, columns, all, name
[].row		number	行のインデックス (0起点)	cell, cells, row, rows
[].rowCount		number	行の数	cells, rows
[].column		number	列のインデックス (0起点)	cell, cells, column, columns
[].columnCount		number	列の数	cells, columns

ドキュメントのインポートとエクスポート

4D View Pro は、複数のドキュメント形式のインポートおよびエクスポートに対応しています:

- .4vp
- .xlsx
- .txt と .csv
- .pdf (エクスポートのみ)

詳細については [VP IMPORT DOCUMENT](#) と [VP EXPORT DOCUMENT](#) の説明を参照ください。

4D View Pro エリアの設定

4D View Pro エリアのプロパティは、プロパティリストを利用して設定することができます。スプレッドシートプロパティはランゲージにより設定します。

フォームエリアプロパティ

オブジェクト名、[変数あるいは式](#)、アピアランス、アクション、イベントなどの [4D View Pro オブジェクトプロパティ](#) は、エリアのプロパティリストを使って設定できます。

ユーザーインターフェースの選択

4D View Pro のフォームエリアで使用するインターフェースは、プロパティリストの [アピアランス](#) から選択できます：



また、`userInterface` および `withFormulaBar` ("ツールバー" インターフェースのみ) の JSONプロパティを使用することもできます。

インターフェースにより、基本的な編集とデータ操作がおこなえます。ユーザーによる編集は、ユーザーによってドキュメントが保存されたときに 4D View Pro オブジェクトに保存されます。

リボン

ツールバー

ツールバー型インターフェースを有効化すると [フォーミュラバーを表示](#) オプションが表示されます。これを選択すると、ツールバーのすぐ下にフォーミュラバーが表示されます。

フォーミュラバーが表示された状態：

機能

リボン型、ツールバー型のいずれのインターフェースでも、関連機能はタブにグループ分けされます：

タブ	アクション	リボン型インターフェース	ツールバー型インターフェース
File	ファイル操作	<input type="radio"/>	
ホーム	テキストの書式など	<input type="radio"/>	<input checked="" type="radio"/>
挿入	アイテムの追加	<input type="radio"/>	<input checked="" type="radio"/>
フォーミュラ	フォーミュラの計算とライブラリ	<input type="radio"/>	<input checked="" type="radio"/>
データ	データ操作	<input type="radio"/>	<input checked="" type="radio"/>
表示	表示の設定	<input type="radio"/>	<input checked="" type="radio"/>
Settings	スプレッドシートの設定	<input type="radio"/>	

フォームイベント

4D View Pro エリアのプロパティリスト内では、以下のフォームイベントが利用可能です。

一部のイベントは（すべてのアクティボオブジェクトで利用可能な）標準のフォームイベントであり、一部は 4D View Pro 専用のフォームイベントです。また一部の標準フォームイベントは、4D View Pro エリアにおいて生成された場合、[FORM Event](#) コマンドが返すオブジェクトに追加の情報を提供します。以下の表は標準イベントと、4D View Pro 専用または追加情報を提供するイベントの一覧です：

標準の 4Dイベント	4D View Pro 専用、または追加情報を返すイベント
On Load	On VP Ready
On Getting Focus	On Clicked
On Losing Focus	On Double Clicked
On Unload	On Header Click
	On After Edit
	On Selection Change
	On Column Resize
	On Row Resize
	On VP Range Changed

シートオプション

4D View Pro シートオプションオブジェクトを使って、4D View Pro エリアの様々なオプションをコントロールすることができます。このオブジェクトは以下のコマンドで操作します：

- [VP SET SHEET OPTIONS](#)
- [VP Get sheet options](#)

シートのアピアランス

プロパティ		タイプ	説明
allowCellOverflow		boolean	セルに収まらないデータを隣の空のセルにはみ出し表示するかどうかを指定します
sheetTabColor		string	シートタブの色を指定するカラー文字列 (例: "red"、"#FFFF00"、"rgb(255,0,0)"、"Accent 5")
frozenlineColor		string	固定化された線の色を指定するカラー文字列 (例: "red"、"#FFFF00"、"rgb(255,0,0)"、"Accent 5")
clipBoardOptions		longint	クリップボードオプション。利用可能な値: <code>vk clipboard paste options all</code> , <code>vk clipboard paste options formatting</code> , <code>vk clipboard paste options formulas</code> , <code>vk clipboard paste options formulas and formatting</code> , <code>vk clipboard paste options values</code> , <code>vk clipboard paste options values and formatting</code>
gridline		object	枠線のオプション
	color	string	枠線の色を表すカラー文字列 (例: "red"、"#FFFF00"、"rgb(255,0,0)"、"Accent 5")
	showVerticalGridline	boolean	垂直の枠線を表示するかどうかを指定します。
	showHorizontalGridline	boolean	水平の枠線を表示するかどうかを指定します。
rowHeaderVisible		boolean	行ヘッダーを表示するかどうかを指定します。
colHeaderVisible		boolean	列ヘッダーを表示するかどうかを指定します。
rowHeaderAutoText		longint	行ヘッダーが文字を表示するか、数字を表示するか、あるいは空かを指定します。利用可能な値: <code>vk header auto text blank</code> , <code>vk header auto text letters</code> , <code>vk header auto text numbers</code>
colHeaderAutoText		longint	列ヘッダーが文字を表示するか、数字を表示するか、あるいは空かを指定します。利用可能な値: <code>vk header auto text blank</code> , <code>vk header auto text letters</code> , <code>vk header auto text numbers</code>
selectionBackColor		string	シートにおける選択範囲の背景色。(RGBAフォーマット推奨)
selectionBorderColor		string	シートにおける選択範囲の枠線の色。
sheetAreaOffset		object	シートエリアのオフセットオプション
	left	longint	シートの、ホストからの左オフセット
	top	longint	シートの、ホストからの上オフセット

いずれのプロパティも任意です。

シートの保護

シート全体をロック (保護) するには、*isProtected* プロパティを true に設定するだけです。その上で、*locked* セルスタイルプロパティを個別に設定することで、特定のセルをロック解除することができます。

プロパティ		タイプ	説明
isProtected		boolean	シート上で保護状態とされているセルが編集可能かどうかを指定します。
protectionOptions		object	ユーザーにより編集可能な要素を指定します。null の場合、protectionOptions パラメーターはリセットされます。
	allowSelectLockedCells	boolean	ロックされたセルをユーザーが選択できるかどうかを指定します (任意)。デフォルトは true。
	allowSelectUnlockedCells	boolean	ロック解除されたセルをユーザーが選択できるかどうかを指定します (任意)。デフォルトは true。
	allowSort	boolean	ユーザーによるレンジの並べ替えが可能かどうかを指定します (任意)。デフォルトは false。
	allowFilter	boolean	ユーザーによるレンジのフィルタリングが可能かどうかを指定します (任意)。デフォルトは false。
	allowEditObjects	boolean	フローティングオブジェクトをユーザーが編集できるかどうかを指定します (任意)。デフォルトは false。
	allowResizeRows	boolean	ユーザーが行をリサイズできるかどうかを指定します (任意)。デフォルトは false。
	allowResizeColumns	boolean	ユーザーが列をリサイズできるかどうかを指定します (任意)。デフォルトは false。
	allowDragInsertRows	boolean	ユーザーがドラッグ操作で行を挿入できるかどうかを指定します (任意)。デフォルトは false。
	allowDragInsertColumns	boolean	ユーザーがドラッグ操作で列を挿入できるかどうかを指定します (任意)。デフォルトは false。
	allowInsertRows	boolean	ユーザーが行を挿入できるかどうかを指定します (任意)。デフォルトは false。
	allowInsertColumns	boolean	ユーザーが列を挿入できるかどうかを指定します (任意)。デフォルトは false。
	allowDeleteRows	boolean	ユーザーが行を削除できるかどうかを指定します (任意)。デフォルトは false。
	allowDeleteColumns	boolean	ユーザーが列を削除できるかどうかを指定します (任意)。デフォルトは false。

いずれのプロパティも任意です。

セルフォーマット

フォーマットパターン (表示形式) を定義することで、4D View Pro ドキュメントのコンテンツを想定通りに表示することができます。フォーマットは、選択された 4D View Pro の インターフェース を使用するか、VP SET VALUE または VP SET NUM VALUE メソッドを使用して設定します。

4D View Pro には数値、日付、時間、そしてテキスト用のビルトインのフォーマットがありますが、カスタムパターンを作成することで、特殊文字やコードを使ったフォーマットでセルのコンテンツを表示することができます。

たとえば、請求書において VP SET VALUE あるいは VP SET NUM VALUE メソッドを使用して金額を入力している場合、数値の桁数とは関係なく (つまり金額が \$5.00 だとうと \$5,000.00 だとうと) 通貨記号 (\$, €, ¥, など) を同じ位置に整列させたい場合があるかもしれません。この場合、フォーマット文字を使用してパターン (\$* #,##0.00) を指定することで、以下のように表示させることができます:

\$	4,180.00
\$	15.00
\$	200.00
\$	15,600.00
\$	1,672.00

カスタムのフォーマットパターンを作成する場合、データの表示のみが変更されるという点に注意してください。データの値そのものは変わりません。

数値とテキストのフォーマット

数値フォーマットはすべての数値型（例：正の数、負の数、ゼロ）に対して適用されます。

文字	説明	例題
0	ゼロを表示する桁のプレースホルダー	#.00 は 1.1 を 1.10 と表示します。
.	小数点を表示します	0.00 は 1999 を 1999.00 と表示します。
,	数値内に千区切りのカンマを表示します。 数値記号 "#" あるいはゼロに挟まれたカンマがフォーマットに含まれる場合、3行ごとにカンマで区切られます。桁のプレースホルダーの後にあるカンマは、数値を 1000 で割ります。	,0 は 12200000 を 12,200,000 と表示します。
_	アンダーバーに続く文字の幅をスキップします。	_（のようにカッコなどと組み合わせることで、左や右にスペースを追加します。
@	テキストのフォーマット文字。セル内のすべてのテキストにフォーマットを適用します。	"[Red]@" はテキスト値に対して赤のフォントカラーを適用します。
*	列幅いっぱいまで、後に続く文字を繰り返します。	0- は、数値の後にセルの幅いっぱいまでダッシュを繰り返します。対して、0 をフォーマットの前につけると、先頭に 0 が複数表示されます。
" "	引用符にはさまれたテキストを、解釈せずにそのまま表示します。	"8%" は 8% と表示されます。
%	数値を百分率で表示します。	8% は、.08 として表示されます。
#	追加のゼロを表示しない、桁のプレースホルダー。もしプレースホルダーの数以上に小数点以下の桁数があった場合、それらの数字は丸められます。	#.# は 1.54 を 1.5 として表示します。
?	追加のゼロの分のスペースを残すが、そのゼロは表示しない桁のプレースホルダー。通常、数値を小数点の位置で揃えるために使用されます。	\$?? は最低 2桁のスペースを確保し、1桁の数値が混じっていても 2桁の場所に \$記号が並ぶように指定します。
¥	後に続く文字を表示します。	#.00? は 123 を 123.00? として表示します。
/	数値に対して使用した場合、分数として表記します。テキスト、日付、時刻に対して使用した場合、それらをそのまま表示します。	#/# は .75 を 3/4 として表示します。
[]	条件つき書式を作成します。	[>100][GREEN]#,##0;[<=-100][YELLOW]#,##0;[BLUE]#,##0
E	指数表記のフォーマット。	#E+# - は 1,500,500 を 2E+6 として表示します。
[color]	テキストまたは数値を指定カラーで表示します。	[Green]###.###[Red]-###.###[Blue]

例題

```
// セルの値を $125,571.35 と表示する設定
VP SET VALUE(VP Cell("ViewProArea";3;2);New object("value";125571.35;"format";"_(* #,##0.00_)"))
```

日付と時間のフォーマット

4D View Pro では ISO 8601 の日付と時間パターンを示す以下の定数を提供しています:

定数	値	説明
<code>vk pattern full date time</code>	<code>"fullDateTimePattern"</code>	カレント言語設定における ISO 8601 フォーマットの完全な日付と時間。 アメリカ合衆国でのデフォルトパターン: "ddd, dd MMMM yyyy HH:mm:ss"
<code>vk pattern long date</code>	<code>"longDatePattern"</code>	カレント言語設定における ISO 8601 フォーマットの完全な日付。 アメリカ合衆国でのデフォルトパターン: "ddd, dd MMMM yyyy"
<code>vk pattern long time</code>	<code>"longTimePattern"</code>	カレント言語設定における ISO 8601 フォーマットの時間。 アメリカ合衆国でのデフォルトパターン: "HH:mm:ss"
<code>vk pattern month day</code>	<code>"monthDayPattern"</code>	カレント言語設定における ISO 8601 フォーマットの月日の日付。 アメリカ合衆国でのデフォルトパターン: "MMM dd"
<code>vk pattern short date</code>	<code>"shortDatePattern"</code>	カレント言語設定における省略形の ISO 8601 フォーマットの日付。 アメリカ合衆国でのデフォルトパターン: "MM/dd/yyyy"
<code>vk pattern short time</code>	<code>"shortTimePattern"</code>	カレント言語設定における省略形の ISO 8601 フォーマットの時間。 アメリカ合衆国でのデフォルトパターン: "HH:mm"
<code>vk pattern sortable date time</code>	<code>"sortableDateTimePattern"</code>	カレント言語設定における、並べ替え可能な ISO 8601 フォーマットの日付と時間。 アメリカ合衆国でのデフォルトパターン: "yyyy-'MM'-'dd'T'HH':'mm':'ss"
<code>vk pattern universal sortable date time</code>	<code>"universalSortableDateTimePattern"</code>	カレント言語設定における、UTCを使用した並べ替え可能な ISO 8601 フォーマットの日付と時間。 アメリカ合衆国でのデフォルトパターン: "yyyy-'MM'-'dd HH':'mm':'ss'Z"
<code>vk pattern year month</code>	<code>"yearMonthPattern"</code>	カレント言語設定における ISO 8601 フォーマットの年月。 アメリカ合衆国でのデフォルトパターン: "yyyy MMMM"

例題

```
// セルの値を特定の日付と時間として表示する設定  
VP SET VALUE(VP Cell("ViewProArea";3;9);New object("value";!2024-12-18!); "time";?14:30:10?; "format";vk p
```

カスタムの日付と時間のフォーマット

カレント言語設定における日付と時間のカスタムパターンを作成したい場合、以下のコードを組み合わせて使用することができます：

	コード (文字の大小の区別なし)	説明	例題
日付			(January 1, 2019)
	m	先頭のゼロなしの月表示	1
	mm	先頭のゼロありの月表示	01
	mmm	短い月名	Jan
	mmmm	長い月名	January
	d	先頭のゼロなしの日付表示	1
	dd	先頭のゼロありの日付表示	01
	ddd	短い曜日名	Tue
	dddd	長い曜日名	Tuesday
	yy	短い年表示	19
	yyyy	長い年表示	2019
時間			(2:03:05 PM)
	h	先頭のゼロなしの時間。0-23の値	2
	hh	先頭のゼロありの時間。00-23の値	02
	m	先頭のゼロなしの秒数。0-59の値	3
	mm	先頭のゼロありの秒数。00-59の値	03
	s	先頭のゼロなしの分數。0-59の値	5
	ss	先頭のゼロありの分數。00-59の値	05
	[h]	経過時間の時間数	14 (24 を超える表示も可能)
	[mm]	経過時間の分數	843
	[ss]	経過時間の秒数	50585
	AM/PM	午前/午後。省略された場合、24時間表記が適用されます。	PM

'm' のコードはその位置とパターンに応じて解釈されます。'h' または 'hh' の直後、あるいは 's' または 'ss' の直前に位置する場合には分を表すと解釈されますが、それ以外の場合には月を表すと解釈されます。

追加の記号

上記で説明されている特殊文字とコードに加えて、フォーマットパターンに使用可能な追加の文字と記号があります。これらの追加文字と記号は ¥ または "" を必要とせず、フォーマットパターンの解釈に影響することはありません。これらはパターン内において "そのまま" 表示されます。

文字	説明	例題
+ と -	プラス記号およびマイナス記号	### + ### = ###,###
()	左と右のカッコ記号	(-###.##)
:	コロン	hh:mm:ss
^	キャレット	#^#
'	アポストロフィー	'######
{ }	中カッコ	{###,###,###}
< >	小なり大なり	## > ##
=	等号	# + # = # #
/	フォワードスラッシュ。数値に対して使用した場合、分数として表記します。	mm/dd/yyyy
!	感嘆符	\$###.00!
&	アンパサンド	"Hello" & "Welcome"
~	チルダ	~##
	スペース文字	
€	ユーロ	€###.00
£	英ポンド	£###.00
¥	日本円	¥###.00
\$	ドル記号	\$###.00
¢	セント記号	.00¢

印刷属性

4D View Pro 印刷属性を使って、4D View Pro エリアの印刷に関するすべての項目を管理することができます。これらの属性は以下のコマンドによって操作します:

- [VP SET PRINT INFO](#)
- [VP Get print info](#)

カラム / 行

カラム属性と行属性を使って、カラムと行の始まり、終わり、そして繰り返しについて指定します。

プロパティ	タイプ	説明
columnEnd	longint	セルレンジ内で印刷する最後のカラム。デフォルト値 = -1 (全カラム)
columnStart	longint	セルレンジ内で印刷する最初のカラム。デフォルト値 = -1 (全カラム)
repeatColumnEnd	longint	各ページの左に印刷するカラムのレンジの、最後のカラム。デフォルト値 = -1 (全カラム)
repeatColumnStart	longint	各ページの左に印刷するカラムのレンジの、最初のカラム。デフォルト値 = -1 (全カラム)
repeatRowEnd	longint	各ページの上に印刷する行のレンジの、最後の行。デフォルト値 = -1 (すべての行)
repeatRowStart	longint	各ページの上に印刷する行のレンジの、最初の行。デフォルト値 = -1 (すべての行)
rowEnd	longint	セルレンジ内で印刷する最後の行。デフォルト値 = -1 (すべての行)
rowStart	longint	セルレンジ内で印刷する最初の行。デフォルト値 = -1 (すべての行)

ヘッダー / フッター

ヘッダー属性とフッター属性は、ヘッダー/フッターセクションの左・右・センターのテキストまたは画像を指定するのに使用されます。

プロパティ	タイプ	説明
footerCenter	text	印刷ページのセンターフッターのテキストとフォーマット
footerCenterImage	picture text*	フッターのセンターセクションの画像
footerLeft	text	印刷ページの左フッターのテキストとフォーマット
footerLeftImage	picture text*	フッターの左セクションの画像
footerRight	text	印刷ページの右フッターのテキストとフォーマット
footerRightImage	picture text*	フッターの右セクションの画像
headerCenter	text	印刷ページのセンターへッダーのテキストとフォーマット
headerCenterImage	picture text*	ヘッダーのセンターセクションの画像
headerLeft	text	印刷ページの左ヘッダーのテキストとフォーマット
headerLeftImage	picture text*	ヘッダーの左セクションの画像
headerRight	text	印刷ページの右ヘッダーのテキストとフォーマット
headerRightImage	picture text*	ヘッダーの右セクションの画像

* テキスト型を使用する場合には、画像のファイルパス（絶対パスまたは相対パス）を渡します。相対パスを渡す場合、ファイルはデータベースのストラクチャーファイルの階層に置かれてなければなりません。Windows では、ファイル拡張子も含めて渡します。画像指定に使用するデータ型にかかわらず、4D View Pro エリアには（参照ではなく）画像そのものが保存され、[VP Get print info](#) によって返されます。

特殊文字

以下の特殊文字を使用すると、4D View Pro エリアが印刷される際にヘッダーとフッター内に自動で情報を追加およびフォーマットすることができます。

文字	説明	例題	戻り値
&	エスケープ文字	(以下の例を参照)	
P	カレントページ	printInfo.headerLeft:="これは &P ページ目です"	これは 5 ページ目です
N	ページ数	printInfo.headerLeft:="&N ページあります"	10 ページあります
D	カレント日付 (yyyy/mm/dd フォーマット)	printInfo.headerLeft:="日付は &D です"	日付は 2015/6/19 です
T	現在の時刻	printInfo.headerLeft:="時刻は &T です"	時刻は 16:30:36 です
G	ピクチャー	printInfo.headerLeftImage:=smiley printInfo.headerLeft:="&G"	
S	打ち消し線	printInfo.headerLeft:="&Sこれはテキストです"	これはテキストです
U	下線	printInfo.headerLeft:="&Uこれはテキストです"	<u>これはテキストです</u>
B	太字	printInfo.headerLeft:="&Bこれはテキストです"	これはテキストです
I	イタリック	printInfo.headerLeft:="&Iこれはテキストです"	これはテキストです
"	フォント指定	printInfo.headerLeft:="&"Lucida Console"&14This is text."	This is text.
K	文字カラー指定	printInfo.headerLeft:="&KFF0000これはテキスト です"	これはテキストです
F	ワークブック名	printInfo.headerLeft:="&F"	2019 Monthly Revenue Forecasts
A	スプレッドシート名	printInfo.headerLeft:="&A"	June 2019 revenue forecast

マージン

マージン属性は、印刷時の 4D View Pro エリアのマージンを指定するために使用されます。100分の1インチ単位で表現されます。

プロパティ		タイプ	説明
margin		object	印刷マージン
	top	longint	上部マージン、100分の1インチ単位。デフォルト値 = 75
	bottom	longint	下部マージン、100分の1インチ単位。デフォルト値 = 75
	left	longint	右マージン、100分の1インチ単位。デフォルト値 = 70
	right	longint	左マージン、100分の1インチ単位。デフォルト値 = 70
	headers	longint	ヘッダーのオフセット、100分の1インチ単位。デフォルト値 = 30
	footer	longint	フッターのオフセット、100分の1インチ単位。デフォルト値 = 30

方向

向き属性は、印刷ページレイアウトの方向を指定するのに使用されます。

この属性はレンダリング情報のみを定義します。

プロパティ	タイプ	説明
orientation	longint	ページの向き。とりうる値: <code>vk print page orientation landscape</code> , <code>vk print page orientation portrait</code> (デフォルト)

ページ

ページ属性は、一般的なドキュメント印刷設定を指定するのに使用されます。

プロパティ	タイプ	説明
blackAndWhite	boolean	白黒で印刷します。 デフォルト値 = <code>false</code> 注: PDF はこの属性に影響されません。PDF のカラーはそのままです。
centering	longint	印刷ページ上でコンテンツをどのように中央揃えするかを指定します。とりうる値: <code>vk print centering both</code> , <code>vk print centering horizontal</code> , <code>vk print centering none</code> (デフォルト), <code>vk print centering vertical</code>
firstPageNumber	longint	最初のページに印刷するページ番号。 デフォルト値 = <code>1</code>
pageOrder	longint	ページの印刷順。とりうる値: <code>vk print page order auto</code> (default), <code>vk print page order down then over</code> , <code>vk print page order over then down</code> .
pageRange	text	印刷されるページの範囲
qualityFactor	longint	印刷の品質指定 (1 - 8)。高ければ印刷の質は高くなりますが、印刷のパフォーマンスに影響する可能性があります。 デフォルト値 = <code>2</code>
useMax	boolean	データのあるカラムと行のみが印刷されます。 デフォルト値 = <code>true</code>
zoomFactor	real	印刷ページの拡大/縮小率。 デフォルト値 = <code>1</code>

用紙サイズ

用紙サイズ属性は印刷に使用する用紙の寸法や規格サイズを指定します。用紙サイズを定義するには 2つの方法があります:

- カスタムサイズ - `height` と `width` 属性を使用
- 規格サイズ - `kind` 属性を使用

プロパティ		タイプ	説明
<code>paperSize</code>		object	印刷に使用する用紙の寸法 (<code>height</code> , <code>width</code>) または規格 (<code>kind</code>)
	<code>height</code>	longint	用紙の高さ、100分の1インチ単位
	<code>width</code>	longint	用紙の幅、100分の1インチ単位
	<code>kind</code>	text	用紙の規格サイズの名前 (例: A2, A4, legal, など)。 <code>GET PRINT OPTION</code> によって返されます。 デフォルト値 = "letter"

スケール

スケール属性は印刷の最適化と調整のために使用されます。

プロパティ	タイプ	説明
bestFitColumns	boolean	印刷時、カラムの幅はテキストの最大幅に合うように調整されます。 デフォルト値 = false
bestFitRows	boolean	印刷時、行の高さはテキストの最大高さに合うように調整されます。 デフォルト値 = false
fitPagesTall	longint	最適化印刷時、チェックする垂直方向（縦向き）のページ数。 デフォルト値 = -1
fitPagesWide	longint	最適化印刷時、チェックする水平方向（横向き）のページ数。 デフォルト値 = -1

表示 / 非表示

表示 / 非表示属性は 4D View Pro エリア要素の表示（印刷）状態を指定するのに使用されます。

プロパティ	タイプ	説明
showBorder	boolean	外枠の境界線を印刷します。 デフォルト値 = true
showColumnHeader	longint	カラムヘッダーの印刷設定。とりうる値: <code>vk print visibility hide</code> , <code>vk print visibility inherit (default)</code> , <code>vk print visibility show</code> , <code>vk print visibility show once</code>
showGridLine	boolean	枠線を印刷します。 デフォルト値 = false
showRowHeader	longint	行ヘッダーの印刷設定。とりうる値: <code>vk print visibility hide</code> , <code>vk print visibility inherit (default)</code> , <code>vk print visibility show</code> , <code>vk print visibility show once</code>

ウォーターマーク

ウォーターマーク属性は 4D View Pro エリアに透かしとして、テキストまたは画像を重ねて表示するために使用されます。

プロパティ		タイプ	説明
watermark		collection	ウォーターマーク設定のコレクション。 デフォルト値: undefined
	[].height	longint	ウォーターマークのテキスト/画像の高さ。
	[].imageSrc	picture text*	ウォーターマークのテキスト/画像。
	[].page	text	ウォーターマークが印刷されるページ。 全ページに印刷: "all"。特定のページ: カンマで区切られたページ番号またはページの範囲。例: "1,3,5-12"
	[].width	longint	ウォーターマークのテキスト/画像の幅。
	[].x	longint	ウォーターマークのテキスト/画像の左上端の水平方向の座標
	[].y	longint	ウォーターマークのテキスト/画像の左上端の垂直方向の座標

* テキスト型を使用する場合には、画像のファイルパス（絶対パスまたは相対パス）を渡します。相対パスを渡す場合、ファイルはデータベースのストラクチャーファイルとの階層に置かれてなければなりません。Windows では、ファイル拡張子も含めて渡します。画像指定に使用するデータ型にかかるわざ、4D View Pro エリアには（参照ではなく）画像そのものが保存され、[VP Get print info](#) によって返されます。

スタイルオブジェクト

4D View Pro スタイルオブジェクトとスタイルシートを使用すると、4D View Pro ドキュメントのグラフィカル要素や見た目を管理することができるようになります。

スタイルオブジェクトとスタイルシート

スタイルオブジェクトには、スタイル設定が格納されます。これらはスタイルシートで使用するか、あるいはそのまま使用することができます。スタイルオブジェクトはスタイルシートと組み合わせて使用することもでき、ドキュメントの他の部分に影響を及ぼすことなく個別のセルレンジに異なる設定を指定することもできます。[VP SET CELL STYLE](#) および [VP SET DEFAULT STYLE](#) コマンドでは、スタイルオブジェクトを直接使用することができます。

スタイルシートは、プロパティの組み合わせをスタイルオブジェクトにまとめたもので、それによって 4D View Pro ドキュメントのすべてのセルの見た目を指定します。ドキュメントとともに保存されたスタイルシートを使用して、単一のシート、複数のシート、あるいはワークブック全体に対してプロパティを設定することができます。4D View Pro スタイルシートは作成時に名前が与えられ、この名前はスタイルシートの "name" プロパティに保存されます。これによりスタイルシートの使用が容易になり、また注意深く命名することで、その定義と目的を分かりやすくすることもできます (例: Letterhead_internal、Letterhead_external、など)。

スタイルシートは [VP ADD STYLESHEET](#) コマンドで作成され、[VP SET DEFAULT STYLE](#) あるいは [VP SET CELL STYLE](#) コマンドで適用されます。スタイルシートは [VP REMOVE STYLESHEET](#) コマンドで削除できます。

[VP Get stylesheet](#) コマンドを使用することでスタイルシートのスタイルオブジェクトを取得できます。また、[VP Get stylesheets](#) コマンドを使用して複数のスタイルシートのスタイルオブジェクトのコレクションを取得することもできます。

スタイルオブジェクトプロパティ

例:

```
$style:=New object
	style.hAlign:=vk horizontal align left
	style.font:="12pt papyrus"
	style.backColor:="#E6E6FA" // 薄紫色
	VP SET DEFAULT STYLE("myDoc";$style)
```

背景色と文字色

プロパティ	タイプ	説明	とりうる値
backColor	text	背景色を定義します。	CSSカラー "#rrggbb" シンタックス (推奨シンタックス)、CSSカラー "rgb(r,g,b)" シンタックス (代替シンタックス)、CSSカラーネーム (代替シンタックス)
backgroundImage	picture, text	背景画像を指定します。	直接指定するか、または画像パス (フルパス、またはファイル名のみ) で指定することができます。ファイル名のみを使用する場合、ファイルはデータベースのストラクチャーファイルと同じ階層に置かれている必要があります。指定の方法 (ピクチャーまたはテキスト) に関わらず、ピクチャーはドキュメントとともに保存されます。画像のサイズが大きい場合、ドキュメントのサイズに影響する場合があります。Windows での注意: ファイル拡張子も含める必要があります。
backgroundImageLayout	longint	背景画像のレイアウトを定義します。	<code>vk image layout center, vk image layout none, vk image layout stretch, vk image layout zoom</code>
foreColor	text	文字のカラーを定義します。	CSSカラー "#rrggbb" シンタックス (推奨シンタックス)、CSSカラー "rgb(r,g,b)" シンタックス (代替シンタックス)、CSSカラーネーム (代替シンタックス)

境界線

プロパティ		タイプ	説明	とりうる値
borderBottom, borderLeft, borderRight, borderTop, diagonalDown, diagonalUp		object	それぞれに対応する境界線を定義します。	
	color	text	境界線のカラーを定義します。デフォルト = black	CSSカラー "#rrggb" シンタックス (推奨シンタックス)、CSSカラー "rgb(r,g,b)" シンタックス (代替シンタックス)、CSSカラーネーム (代替シンタックス)
	style	longint	境界線のスタイルを定義します。デフォルト = empty。 null または未定義をとることはできません。	<code>vk line style dash dot</code> , <code>vk line style dash dot dot</code> , <code>vk line style dashed</code> , <code>vk line style dotted</code> , <code>vk line style double</code> , <code>vk line style empty</code> , <code>vk line style hair</code> , <code>vk line style medium</code> , <code>vk line style medium dash dot</code> , <code>vk line style medium dashed</code> , <code>vk line style slanted dash dot</code> , <code>vk line style thick</code>

フォントとテキスト

プロパティ		タイプ	説明	とりうる値
font		text	フォントの特徴を CSS の fontショートハンドで指定します ("font-style font-variant font-weight font-size/line-height font-family")。例: "14pt Century Gothic"。フォントサイズ (font-size) とフォントファミリー (font-family) の値は必須です。その他の値が省略された場合には、そのデフォルト値が使用されます。注: フォント名にスペースが含まれる場合、その名前は引用符 ("") で括られる必要があります。	CSS fontショートハンド。 4D ではフォントの特徴をオブジェクトとして管理するためのユーティリティコマンドを提供しています: VP Font to object および VP Object to font
formatter		text	値や日時に対するパターン	数値/テキスト/日付/時間フォーマット、特殊文字など。 セルフォーマット 参照。
isVerticalText		boolean	テキストの向きを指定します。	true = 縦方向のテキスト, false = 横方向のテキスト
labelOptions		object	セルラベルのオプションを定義します (ウォーターマークオプション)	
	alignment	longint	セルラベルの位置を指定します。任意プロパティです。	<code>vk label alignment top left</code> , <code>vk label alignment bottom left</code> , <code>vk label alignment top center</code> , <code>vk label alignment bottom center</code> , <code>vk label alignment top right</code> , <code>vk label alignment bottom right</code>
	visibility	longint	セルラベルの表示状態を指定します。任意プロパティです。	<code>vk label visibility auto</code> , <code>vk label visibility hidden</code> , <code>vk label visibility visible</code>

プロパティ	foreColor	text	説明 テキストのカラーを定義します。任意プロパティです。	CSSカラーコード ("#rrggbbaa" シンタクス (推奨シンタクス)、CSSカラー "rgb(r,g,b)" シンタクス (代替シンタクス)、CSSカラーネーム (代替シンタクス)
	font	text	フォントの特徴を CSS の fontショートハンドで指定します ("font-style font-variant font-weight font-size/line-height font-family")。フォントサイズ (font-size) とフォントファミリー (font-family) の値は必須です。	
textDecoration		longint	テキストに追加する装飾を指定します。	<code>vk text decoration double underline, vk text decoration line through, vk text decoration none, vk text decoration overline, vk text decoration underline</code>
textIndent		longint	テキストのインデントを定義します。1 = 8ピクセル	
textOrientation		longint	セル内のテキストの回転角度を定義します。-90 から 90 の数値	
watermark		text	ウォーターマーク (セルラベル) のコンテンツを定義します。	
wordWrap		boolean	テキストを折り返すかどうかを指定します。	true = テキストを折り返す、false = テキストを折り返さない

レイアウト

プロパティ	タイプ	説明	とりうる値
cellPadding	text	セルのパッディングを定義します	
hAlign	longint	セルコンテンツの水平方向の揃え方を定義します	<code>vk horizontal align center, vk horizontal align general, vk horizontal align left, vk horizontal align right</code>
locked	boolean	セルの保護状態を指定します。ただし、シートの保護が有効化されている場合にのみ利用可能である点に注意してください。	true = ロックされている、false = ロック解除
shrinkToFit	boolean	セルのコンテンツが縮小されるかどうかを指定します。	true = コンテンツ縮小、false = 縮小なし
tabStop	boolean	Tabキーを使用してセルにフォーカスできるかどうかを指定します。	true = Tabキーでフォーカス可、false = Tabキーでフォーカス不可
vAlign	longint	セルコンテンツの垂直方向の揃え方を定義します	<code>vk vertical align bottom, vk vertical align center, vk vertical align top</code>

スタイル情報

プロパティ	タイプ	説明
name	text	スタイルの名前を定義します。
parentName	text	カレントスタイルの元となっているスタイルを指定します。まず親スタイルの値が適用され、次にカレントスタイルの値が適用されます。カレントスタイルの変更は親スタイルには反映されません。これはスタイルシートを使用している時の利用可能です。

4D View Pro オブジェクト

4D View Pro オブジェクト はスプレッドシートの中身をすべて保存します。このオブジェクトは 4D View Pro によって自動的に管理されます。 [VP IMPORT FROM OBJECT](#) および [VP Export to object](#) メソッドによって、このオブジェクトを設定したり取得したりできます。

このオブジェクトには次のプロパティが含まれます:

プロパティ	値の型	説明
version	Longint	内部コンポーネントのバージョン
dateCreation	Timestamp	作成日
dateModified	Timestamp	最終更新日
meta	Object	4Dデベロッパー専用の任意コンテンツ
spreadJS	Object	4D View Pro コンポーネント専用

4D View Pro フォームオブジェクト変数

The 4D View Pro フォームオブジェクト変数は、4D View Pro フォームエリアに紐づけられた オブジェクト 変数のことです。この変数は、4D View Pro オブジェクトが使用する情報を管理します。

4D View Pro フォームオブジェクト変数は、情報目的（例: デバッグなど）のためだけに存在します。どのような状況においてもこれを編集してはいけません。

このオブジェクトには次のプロパティが含まれます:

プロパティ	値の型	説明
ViewPro.area	Text	4D View Pro エリア名
ViewPro.callbacks	Object	インポートやエクスポートなど、コールバックが必要とするコマンドが使用するための一時的な情報を作成します。
ViewPro.commandBuffers	Collection	メソッドによって呼び出されるコマンドを順番に保存し、メソッド終了時、あるいはコマンドが値を返した時、あるいは VP FLUSH COMMANDS が呼び出された時に、それらのコマンドを（個別ではなく）バッチとして実行します。この機構によって、送信されるリクエスト数が抑えられ、パフォーマンスが向上します。
ViewPro.events	Object	イベントリスト。
ViewPro.formulaBar	Boolean	フォーミュラバーが表示されているかどうかを示します。 "toolbar" インターフェースにおいてのみ利用可能です。
ViewPro.initiated	Boolean	4D View Pro エリアが初期化されたかどうかを示します（ On VP Ready 参照）。
ViewPro.interface	Text	ユーザーインターフェースのタイプを指定します: "ribbon"、"toolbar"、"none"。

式と関数

フォーミュラの使い方

スプレッドシートのフォーミュラとは、セルの値を計算する式のことです。

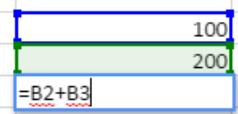
フォーミュラの入力

4D View Pro エリアにフォーミュラを入力するには:

1. フォーミュラを入力するセルを選択します。
2. = (等号) を入力します。
3. フォーミュラを入力し、Enterキーを入力します。

フォーミュラ入力の際には、次のショートカットを使うことができます:

- フォーミュラにセル参照を入力する代わりに、参照するセルをクリックします:



- 入力したい関数の頭文字を入力します。すると、利用可能な関数と参照の一覧がポップアップメニューに表示され、必要な要素を選択することができます:

また、命名フォーミュラを作成すると、その名前で呼び出すことが可能です。この場合、[VP ADD FORMULA NAME](#) コマンドを使用して命名フォーミュラを入力します。

演算子とオペランド

すべてのフォーミュラはオペランドと演算子から成り立っています:

- 演算子: 後述の [値と演算子](#) 参照。
- オペランド は、いくつかのカテゴリーに分けられます:
 - [値](#) (5つのデータ型がサポートされています)
 - [他のセルへの参照](#) (相対参照、絶対参照、ミックス参照、あるいは名前での参照)
 - [標準のスプレッドシートファンクション \(関数\)](#)
 - 4Dフォーミュラに基づく [4Dファンクション](#) (4D変数、フィールド、メソッド、コマンド、式が利用可能です)

値と演算子

4D View Pro は 5つのデータ型をサポートします。それぞれのデータ型について、特定の定数と演算子がサポートされています。

データ型	値	演算子
数値	1.2 1.2 E3 1.2E-3 10.3x	+ (加法) - (減法) * (乗法) / (除法) ^ (べき乗、数値を自身に対してかける回数) % (パーセント -- 演算子の前の数値を100で割る)
日付	10/24/2017	+ (日付 + 日数 -> 日付) + (日付 + 時間 -> 日付 + その日の時間) - (日付 - 日数 -> 日付) - (日付 - 日付 -> 2つの日付間の日数)
時間	10:12:10	経過時間演算子: + (加法) - (減法) * (経過時間 * 数値 -> 経過時間) / (経過時間 / 数値 -> 経過時間)
文字列	'Sophie' または "Sophie"	& (連結)
布尔	TRUE または FALSE	-

比較演算子

同じ型の 2つのオペランドに対して以下の演算子を使用することができます:

演算子	比較
=	等しい
<>	異なる
>	大きい
<	小さい
>=	以上
<=	以下

演算子の優先順位

以下は演算子を優先度順に並べたものです (優先度が高い方が上):

演算子	説明
()	カッコ (グループ化)
-	否定
+	追加
%	パーセント
^	べき乗
* と /	乗法と除法
+ と -	加法と減法
&	連結
= > < >= <= <>	比較

セル参照

フォーミュラ (式) は多くの場合、セルアドレスを使って他のセルを参照します。このような式を他のセルにコピーすることも可能です。たとえば、C8 に入力された以下の式では、すぐ上にある 2つのセルの値を足して結果を表示します。

= C6 + C7

この式はセル C6 と C7 を参照します。つまり、式内で使用する値については他のセルを参照するよう、4D View Pro に指示しています。

こういった式を別のセルにコピーあるいは移動させた場合、式内のセルアドレスは参照の仕方によって連動して変化したり変化しなかったりします。

- 参照が変化するものは 相対参照 と呼ばれ、式が入っているセルを基準として、相対的にどれだけ上下左右に離れているかでセルを参照します。
- 常に特定のセルを参照するものは 絶対参照 と呼ばれます。
- また、複合参照といって、特定の行あるいはカラムに参照を固定することもできます。

参照の記法

セル座標 (C5 など) のみを使用すると、4D View Pro はそれを相対参照として解釈します。文字と数字の前に \$ (ドル) 記号を入れることで、参照を絶対参照にすることができます (例: \$C\$5)。

ドル記号を文字あるいは数字の前だけに入れることで相対参照と絶対参照を組み合わせることができます (例: \$C5 、 C\$5)。複合参照を使用すると、行かカラムを絶対参照として指定しながら、もう片方については相対的に参照させることができます。

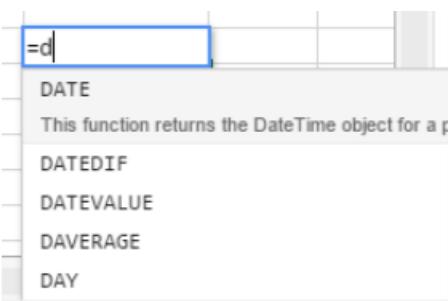
素早くかつ正確に絶対参照を指定するには、セルを命名し、その名前をセルアドレス代わりに使用する方法があります。命名セルへの参照は常に絶対参照です。命名セルまたは命名セルレンジを作成・編集するには、[VP ADD RANGE NAME](#) メソッドを使用します。

以下の表は異なる記法による効果をまとめたものです:

例題	参照タイプ	説明
C5	相対	参照は相対的にセル C5 を指しており、参照が最初に使用されたセルが位置の基準となります。
\$C\$5	絶対	参照は絶対参照です。どこで使用されるかにかかわらず、常にセル C5 を参照します。
\$C5	Mixed	常にカラム C を参照しますが、行の参照は相対的で、参照が最初に使用されたセルを基準に決まります。
C\$5	Mixed	常に 5行目を参照しますが、カラムの参照は相対的で、参照が最初に使用されたセルを基準に決まります。
セル名	絶対	参照は絶対参照です。どこで使用されるかにかかわらず、常に 命名されたセルまたはレンジ を参照します。

ビルトインファンクション

スプレッドシートのファンクションとは、セルの値を計算する規定のフォーミュラのことです、関数とも呼ばれます。入力したい関数の頭文字を入力すると、利用可能な関数と参照の一覧がポップアップメニューに表示され、必要な要素を選択することができます:



詳細や例題については [SpreadJS's extented list of functions](#) (英文) を参照ください。

4Dファンクション

4D View Pro では、[4Dフォーミュラ](#) を実行する 4Dカスタムファンクション を定義・使用することができます。4Dカスタムファンクションを使用すると、4D View Pro ドキュメントの可能性が広がり、4Dデータベースとの強力な連携が可能になります。

4Dカスタムファンクションを使用すると、4D View Pro のフォーミュラ内で以下が利用可能になります:

- 4Dプロセス変数
- フィールド
- プロジェクトメソッド
- 4Dランゲージコマンド
- または、有効な 4D式

4Dカスタムファンクションは、4D View Pro エリアから [引数](#) を受け取り、値を返すことができます。

これらのカスタムファンクションは [VP SET CUSTOM FUNCTIONS](#) メソッドを使って宣言します。例:

```
o:=New object

// 4D View Pro におけるファンクション名: "DRIVERS_LICENCE"
$o.DRIVERS_LICENCE:=New object

// プロセス変数
$o.DRIVERS_LICENCE.formula:=Formula(DriverLicence)

// テプルフィールド
$o.DRIVERS_LICENCE.formula:=Formula([Users]DriverLicence)

// プロジェクトメソッド
$o.DRIVERS_LICENCE.formula:=Formula(DriverLicenceState)

// 4Dコマンド
$o.DRIVERS_LICENCE:=Formula(Choose(DriverLicence; "Obtained"; "Failed"))

// 4D式と引数
$o.DRIVERS_LICENCE.formula:=Formula(ds.Users.get($1).DriverLicence)
$o.DRIVERS_LICENCE.parameters:=New collection
$o.DRIVERS_LICENCE.parameters.push(New object("name"; "ID"; "type"; Is longint))
```

[4D View Pro: Use 4D formulas in your spreadsheet \(英文ブログ記事\)](#) も参照ください。

Hello World 例題

4Dプロジェクトメソッドを使って、4D View Pro エリアのセルに "Hello World" と表示させます。

1. "myMethod" プロジェクトメソッドを作成し、次のコードを書きます:

```
#DECLARE->$hw Text
$hw:="Hello World"
```

2. 4D View Pro エリアが設置されたフォームを開く前に、以下のコードを実行します:

```
Case of
:(Form event code=On Load)
  var $o : Object
  $o:=New object
  // "myMethod" メソッドから "vpHello" ファンクションを定義します
  $o.vpHello:=New object
  $o.vpHello.formula:=Formula(myMethod)
  VP SET CUSTOM FUNCTIONS("ViewProArea";$o)
End case
```

3. 4D View Pro エリアのセルに次を入力します:

```
=VPHELLO()
```

すると、4D によって "myMethod" が呼び出され、セルの表示は次のようにになります:

```
Hello World
```

引数

プロジェクトのメソッドを呼び出す 4Dファンクションには、以下のシンタックスで引数を渡すことができます:

```
=METHODNAME(param1,param2,...,paramN)
```

methodName はこれらの引数を \$1, \$2...\$N に受け取ります。

引数を渡さない場合でも () の使用は必須です:

```
=METHODWITHOUTNAME()
```

VP SET CUSTOM FUNCTIONS メソッドを使用して宣言したファンクションの *parameters* コレクションを使って、引数の名前、型、数を宣言することができます。オプションとして、*minParams* および *maxParams* プロパティにより、ユーザーから渡される引数の数を制御することができます。

サポートされている引数の型の詳細については、[VP SET CUSTOM FUNCTIONS](#) メソッドの説明を参照ください。

引数を宣言していない場合には、メソッドに値を順番に渡すことができ (\$1, \$2... に受け取られます)、それらの型は自動的に変換されます。

jstype の日付は、4Dコードでは 2つのプロパティを持つ [オブジェクト](#) として渡されます:

|プロパティ| 型 | 説明 | |---|---|---| |value| Date| 日付 | |time| Real| 時間 (秒単位)|

4Dプロジェクトメソッドは、\$0 を介して 4D View Pro のセルフォーミュラに値を返すこともできます。以下のデータ型の戻り値がサポートされています:

- [テキスト](#) (4D View Pro 内で文字列に変換)
- [実数/倍長整数](#) (4D View Pro 内で数値に変換)
- [日付](#) (4D View Pro 内で JS日付型に変換 - 時間、分、秒 = 0)
- [時間](#) (4D View Pro 内で JS日付型に変換 - 日付は基準日、つまり 1899年12月30日)
- [ブール](#) (4D View Pro 内でブールに変換)
- [ピクチャー](#) (jpg,png,gif,bmp,svg, その他のタイプは png に変換) の場合、URI (data:image/png;base64,xxxx) が作成され、フォーミュラを実行した 4D View Pro のセルにおいて背景として使用されます。
- 次の 2つのプロパティを持つ [オブジェクト](#) (日付と時間の受け渡しを可能にします):

プロパティ	タイプ	説明
value	日付	日付値
time	実数	数値 (秒単位)

4Dメソッドが何も返さない場合は、自動的に空の文字列が返されます。

以下の場合は、4D View Pro セルにエラーが返されます:

- 4Dメソッドが上記以外のデータ型を返した場合。
- 4Dメソッドの実行中にエラーが発生した場合 (ユーザーが "中止" ボタンをクリックした場合)。

例題

```

var $o : Object

$o.BIRTH_INFORMATION:=New object
$o.BIRTH_INFORMATION.formula:=Formula(BirthInformation)
$o.BIRTH_INFORMATION.parameters:=New collection
$o.BIRTH_INFORMATION.parameters.push(New object("name";"First name";"type";Is text))
$o.BIRTH_INFORMATION.parameters.push(New object("name";"Birthday";"type";Is date))
$o.BIRTH_INFORMATION.parameters.push(New object("name";"Time of birth";"type";Is time))
$o.BIRTH_INFORMATION.summary:="渡された情報に基づきフォーマットされた文字列を返します"

VP SET CUSTOM FUNCTIONS("ViewProArea"; $o)

```

互換性

4D View Pro エリアでフィールドやメソッドをファンクションとして宣言する場合、代わりの方法も利用可能です。これらの方法は、互換性のために維持されており、特定のケースで使用することができます。しかしながら、`VP SET CUSTOM FUNCTIONS` の使用が推奨されます。

仮想ストラクチャーを使ったフィールド参照

4D View Pro では、データベースの仮想ストラクチャーを使用して 4D フィールドを参照することができます。つまり、*引数とともに `SET TABLE TITLES` や `SET FIELD TITLES` コマンドで宣言されている場合です。この代替方法は、アプリケーションがすでに仮想ストラクチャーに依存している場合に便利です（そうでない場合は、`VP SET CUSTOM FUNCTIONS` の使用が推奨されます）。

警告: 仮想ストラクチャーと `VP SET CUSTOM FUNCTIONS` を同時に使用することはできません。`VP SET CUSTOM FUNCTIONS` が呼び出されると、4D View Pro エリアは `SET TABLE TITLES` や `SET FIELD TITLES` コマンドに基づく機能を無視します。

要件

- フィールドは、データベースの仮想ストラクチャーに属していること。つまり、*引数とともに `SET TABLE TITLES` や `SET FIELD TITLES` コマンドで宣言されていないことはなりません。
- テーブルとフィールド名は ECMA 準拠であること（[ECMA Script standard](#) 参照）。
- フィールドの型が 4D View Pro でサポートされていること（前述参照）。

準拠していないフィールドがフォーミュラに呼び出されると、4D View Pro セルにエラーが返されます。

フォーミュラでの仮想フィールドの使用

フォーミュラ内に仮想フィールドへの参照を挿入するには、次のシンタックスを使います：

```
TABLENAME_FIELDNAME()
```

たとえば、"People" テーブルの "Name" フィールドを仮想ストラクチャーで宣言した場合、以下のようなファンクションを呼び出すことができます：

```
=PEOPLE_NAME()
=LEN(PEOPLE_NAME())
```

フィールドが [4Dメソッド] と同名の場合は、フィールド名が優先されます。

例題

4D の仮想フィールドを使用して、4D View Pro エリアのセル内に人の名前を表示します：

- "Employee" テーブルと "L_Name" フィールドを作成します：

Employee	
ID	2 ¹²
L_Name	A
F_Name	A

2. 次のコードを実行して、仮想ストラクチャーを初期化します:

```
ARRAY TEXT($tableTitles;1)
ARRAY LONGINT($tableNum;1)
$tableTitles{1}:="Emp"
$tableNum{1}:=2
SET TABLE TITLES($tableTitles;$tableNum;*)

ARRAY TEXT($fieldTitles;1)
ARRAY LONGINT($fieldNum;1)
$fieldTitles{1}:="Name"
$fieldNum{1}:=2 // ラストネーム
SET FIELD TITLES([Employee];$fieldTitles;$fieldNum;*)
```

3. 4D View Pro エリアのセルに "=e" と入力します:

	A	B	C	D	E
1	=e				
2	EDATE				
3	EFFECT				
4	EMP_NAME				
5	ENCODEURL				

4. (Tabキーを使用して) EMP_NAME を選択し、閉じる ")" を入力します。

	A	B
1	=EMP_NAME()	

5. セルを確定すると、カレントの従業員の名前が表示されます:

	A	B
1	Smith	

[Employee] テーブルはカレントレコードを持っている必要があります。

許可されたメソッドの宣言

4D View Pro のフォーミュラ内で呼び出すには、プロジェクトメソッドは以下の条件を満たしている必要があります:

- 許可されている: [VP SET ALLOWED METHODS](#) によって明示的に宣言されていること。
- 実行可能: メソッドがホストデータベースに属している、あるいはロードされたコンポーネントに属しており当該メソッドの "コンポーネントとホストデータベース間で共有" オプションが有効化されていること ([プロジェクトメソッドの共有](#) 参照)。
- 既存の 4D View Pro ファンクションと 競合していない: 4D View Pro ビルトインファンクションと同じ名前のプロジェクトメソッドを呼び出した場合、ファンクションの方が呼び出されます。

要件

4D View Pro フォーミュラ内で呼び出すには、プロジェクトメソッドは以下の条件を満たしている必要があります:

- 許可されている: [VP SET ALLOWED METHODS](#) によって明示的に宣言されていること。
- 実行可能: メソッドがホストデータベースに属している、あるいはロードされたコンポーネントに属しており当該メソッドの "コンポーネントとホストデータベース間で共有" オプションが有効化されていること ([プロジェクトメソッドの共有](#) 参照)。
- 既存の 4D View Pro ファンクションと 競合していない: 4D View Pro ビルトインファンクションと同じ名前のプロジェクトメソッドを呼び出した場合、ファンクションの方が呼び出されます。

[VP SET CUSTOM FUNCTIONS](#) および [VP SET ALLOWED METHODS](#) コマンドのいずれもがセッション中に実行されていない場合、4D View Pro カスタムファンクションには 4D の汎用的な [SET ALLOWED METHODS](#) コマンドで許可されたメソッドが使用できます。この場合、プロジェクトメソッド名は JavaScript の字句文法に則ってなければなりません ([ECMA Script standard](#) 参照)。ストラクチャー設定のグローバルなフィルタリングオプション (セキュリティページ > データアクセス権) はいずれの場合でも無視されます。

メソッド一覧

警告: このページのコマンドはスレッドセーフではありません。

A - C - D - E - F - G - I - M - N - O - P - R - S

A

VP ADD FORMULA NAME

VP ADD FORMULA NAME (*vpAreaName* : Text ; *vpFormula* : Text ; *name* : Text { ; *options* : Object })

引数	タイプ	説明
<i>vpAreaName</i>	Text	-> 4D View Pro フォームオブジェクト名
<i>vpFormula</i>	Text	-> 4D View Pro フォーミュラ
<i>name</i>	Text	-> フォーミュラの名称
<i>options</i>	Object	-> 命名フォーミュラのオプション

説明

VP ADD FORMULA NAME コマンドは、開いているドキュメント内において命名されたフォーミュラを作成、または編集します。

このコマンドで作成された命名フォーミュラはドキュメントとともに保存されます。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

vpFormula には、命名する 4D View Pro フォーミュラを渡します。フォーミュラシンタックスの詳細については、[式と関数](#) ページを参照ください。

名前をつけたいレンジを *rangeObj* に、新しいレンジの名前は *name* に渡します。同じスコープ内で名前が既に使用されている場合、新しい命名レンジは既存のものを上書きします。ただし異なるスコープであれば同じ名前を使用することができます (以下参照)。

options 引数には、命名フォーミュラの追加プロパティを格納したオブジェクト型を渡すことができます。以下のオブジェクトプロパティがサポートされています:

プロパティ	タイプ	説明
<i>scope</i>	Number	フォーミュラのスコープ。シートのインデックス (0起点) を渡すか、あるいは以下の定数を使用することができます: <ul style="list-style-type: none">● <code>vk current sheet</code>● <code>vk workbook</code> スコープは、フォーミュラ名が特定のワークシートに限定されたローカル (<i>scope</i> = シートのインデックス または <code>vk current sheet</code>) なものか、あるいはワークブック全体で使用できるグローバル (<i>scope</i> = <code>vk workbook</code>) なものかを決定します。
<i>comment</i>	Text	命名フォーミュラに割り当てられたコメント

例題

```
VP ADD FORMULA NAME("ViewProArea";"SUM($A$1:$A$10)";"Total2")
```

参照

[Cell references](#)

[VP ADD RANGE NAME](#)

[VP Get formula by name](#)

[VP Get names](#)

VP ADD RANGE NAME

`VP ADD RANGE NAME (rangeObj : Object ; name : Text { ; options : Object })`

引数	タイプ		説明
rangeObj	Text	->	レンジオブジェクト
name	Text	->	フォーミュラの名称
options	Object	->	命名フォーミュラのオプション

説明

`VP ADD RANGE NAME` コマンドは、開いているドキュメント内に命名レンジを作成、または編集します。

このコマンドで作成された命名レンジはドキュメントとともに保存されます。

name 引数には、新しいフォーミュラの名前を渡します。同じスコープ内で名前が既に使用されている場合、新しい命名フォーミュラは既存のものを上書きします。ただし異なるスコープであれば同じ名前を使用することができます（以下参照）。

options 引数には、命名レンジの追加プロパティを格納したオブジェクト型を渡すことができます。以下のオブジェクトプロパティがサポートされています：

プロパティ	タイプ	説明
scope	Number	レンジのスコープ。シートのインデックス（0起点）を渡すか、あるいは以下の定数を使用することができます： ● <code>vk current sheet</code> ● <code>vk workbook</code> スコープは、レンジ名が特定のワークシートに限定されたローカル（ <code>scope = シートのインデックス</code> または <code>vk current sheet</code> ）なものか、あるいはワークブック全体で使用できるグローバル（ <code>scope = vk workbook</code> ）ものかを決定します。
comment	Text	命名レンジに割り当てられたコメント

- 命名レンジの実態は、座標を格納した命名フォーミュラです。`VP ADD RANGE NAME` を使うと簡単に命名レンジの作成ができますが、`VP ADD FORMULA NAME` コマンドで命名レンジを作成することもできます。
- 命名レンジを定義するフォーミュラは、`VP Get formula by name` コマンドで取得することができます。

例題

あるセルレンジに対して命名レンジを作成します：

```
$range:=VP Cell("ViewProArea";2;10)
VP ADD RANGE NAME($range;"Total1")
```

参照

[VP ADD FORMULA NAME](#)

[VP Get formula by name](#)

[VP Get names](#)

[VP Name](#)

VP ADD SELECTION

VP ADD SELECTION (*rangeObj* : Object)

引数	タイプ		説明
<i>rangeObj</i>	Text	->	レンジオブジェクト

説明

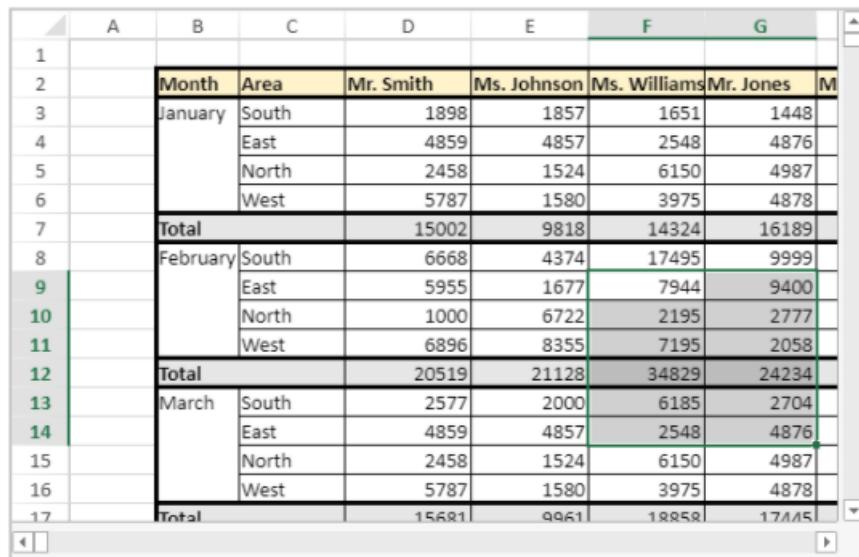
VP ADD SELECTION コマンドは、指定されたセルを、現在選択されているセル範囲に追加します。

rangeObj には、カレントセレクションに追加するセルのレンジオブジェクトを渡します。

アクティブセルは変更されません。

例題

以下のようにセルが選択されている場合:

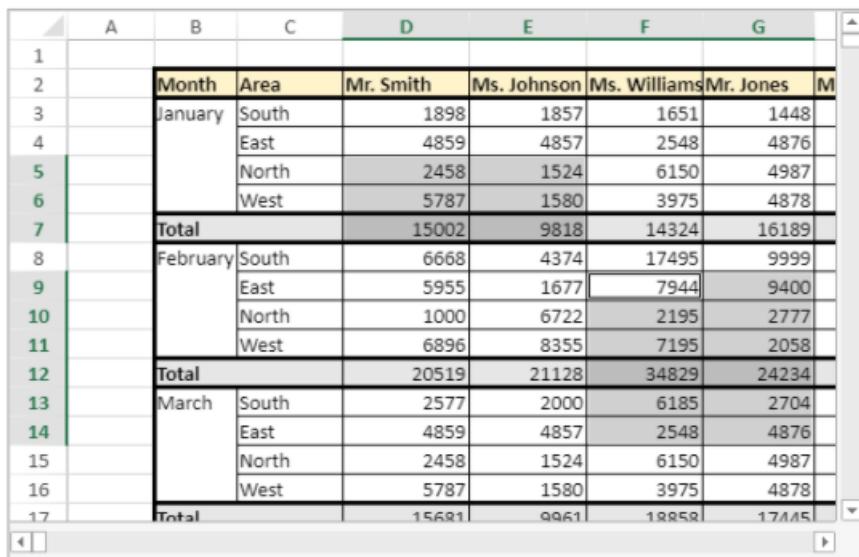


A	B	C	D	E	F	G
1						
2	Month	Area	Mr. Smith	Ms. Johnson	Ms. Williams	Mr. Jones
3	January	South	1898	1857	1651	1448
4		East	4859	4857	2548	4876
5		North	2458	1524	6150	4987
6		West	5787	1580	3975	4878
7		Total	15002	9818	14324	16189
8	February	South	6668	4374	17495	9999
9		East	5955	1677	7944	9400
10		North	1000	6722	2195	2777
11		West	6896	8355	7195	2058
12		Total	20519	21128	34829	24234
13	March	South	2577	2000	6185	2704
14		East	4859	4857	2548	4876
15		North	2458	1524	6150	4987
16		West	5787	1580	3975	4878
17		Total	15681	9961	18858	17445

以下のコードを実行すると、指定したセルを選択範囲に追加します:

```
$currentSelection:=VP_Cells("myVPArea";3;4;2;3)
VP_ADD_SELECTION($currentSelection)
```

結果:



A	B	C	D	E	F	G
1						
2	Month	Area	Mr. Smith	Ms. Johnson	Ms. Williams	Mr. Jones
3	January	South	1898	1857	1651	1448
4		East	4859	4857	2548	4876
5		North	2458	1524	6150	4987
6		West	5787	1580	3975	4878
7		Total	15002	9818	14324	16189
8	February	South	6668	4374	17495	9999
9		East	5955	1677	7944	9400
10		North	1000	6722	2195	2777
11		West	6896	8355	7195	2058
12		Total	20519	21128	34829	24234
13	March	South	2577	2000	6185	2704
14		East	4859	4857	2548	4876
15		North	2458	1524	6150	4987
16		West	5787	1580	3975	4878
17		Total	15681	9961	18858	17445

参照

[VP Get active cell](#)
[VP Get selection](#)
[VP RESET SELECTION](#)
[VP SET ACTIVE CELL](#)
[VP SET SELECTION](#)
[VP SHOW CELL](#)

VP ADD SHEET

VP ADD SHEET (*vpAreaName* : Text)
VP ADD SHEET (*vpAreaName* : Text ; *index* : Integer)
VP ADD SHEET (*vpAreaName* : Text ; *index* : Integer ; *name* : Text)

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro フォームオブジェクト名
<i>index</i>	Integer	->	新しいシートのインデックス
<i>name</i>	Text	->	シート名

説明

VP ADD SHEET コマンドは、*vpAreaName* にロードされているドキュメントにシートを挿入します。

vpAreaName には、4D View Pro エリアの名前を渡します。

index 引数として、新しいシートのインデックスを渡します。渡した *index* 引数が 0 以下だった場合、コマンドは新しいシートを先頭に挿入します。
index 引数がシートの総数より多い場合、コマンドは既存のシートの後に新しいシートを挿入します。

インデックスは 0 起点です。

name 引数として、新しいシートの名前を渡します。新しい名前には、次の文字を含めることはできません: *, :, [,], ?, \, /

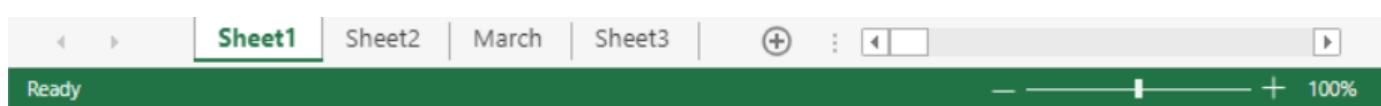
例題

ドキュメントには現在 3つのシートがあります:



新しいシートを 3つ目の位置 (インデックスは 2) に挿入し、名前を "March" にします:

```
VP ADD SHEET("ViewProArea";2;"March")
```



参照

[VP REMOVE SHEET](#)

VP ADD SPAN

VP ADD SPAN (*rangeObj* : Object)

引数	タイプ		説明
rangeObj	Object	->	レンジオブジェクト

説明

VP ADD SPAN コマンドは、*rangeObj* に渡したセルを单一のセルに結合します。

rangeObj には、セルのレンジオブジェクトを渡します。レンジ内のセルは結合され、複数のカラム/行にまたがる大きなセルが作成されます。複数のセルレンジを渡すことで、一度に複数の結合セルを作成することもできます。ただし、セルレンジが重なった場合、最初のセルレンジのみが使用されます。

- 結合セルでは、左上端セルのデータのみが表示されます。他のセルのデータは結合が解除されるまで非表示になります。
- 結合セル内の非表示データは、フォーミュラを使用することでアクセス可能です（フォーミュラは左上端セルから始まります）。

例題

"First quarter" セルと "Second quarter" セルを、それぞれ右 2つのセルと結合し、"South area" セルは下 2つのセルと結合します：

		First quart			Second qu		
		Jan	Feb	Mar	Apr	May	Jun
South area	John						
	Sahra						
	Dixie						

```
// "First quarter" レンジ
$q1:=VP Cells("ViewProArea";2;3;3;1)

// "Second quarter" レンジ
$q2:=VP Cells("ViewProArea";5;3;3;1)

// "South area" レンジ
$south:=VP Cells("ViewProArea";0;5;1;3)

VP ADD SPAN(VP Combine ranges($q1;$q2;$south))
```

		First quarter			Second quarter		
		Jan	Feb	Mar	Apr	May	Jun
South area	John						
	Sahra						
	Dixie						

参照

[4D View Pro Range Object Properties](#)

[VP Get spans](#)

[VP REMOVE SPAN](#)

VP ADD STYLESHEET

VP ADD STYLESHEET (*vpAreaName* : Text ; *styleName* : Text ; *styleObj* : Object { ; *scope* : Integer })

引数	タイプ		説明
vpAreaName	Text	->	4D View Pro フォームオブジェクト名
styleName	Text	->	スタイルの名前
styleObj	Object	->	属性設定を定義するオブジェクト
scope	Integer	->	ターゲットのスコープ (デフォルト = カレントシート)

説明

VP ADD STYLESHEET コマンドは、開いているドキュメント内にて、styleName 引数で指定したスタイルシートを、styleObj 引数のプロパティの組み合わせに基づいて作成または変更します。同じ名前とスコープを持つスタイルシートがドキュメント内にすでに存在する場合、このコマンドはそれを新しい値で上書きします。

このコマンドで作成されたスタイルシートはドキュメントとともに保存されます。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

styleName 引数には、スタイルシートの名前を渡します。同じスコープ内で名前が既に使用されている場合、新しいスタイルシートは既存のものを上書きします。ただし異なるスコープであれば同じ名前を使用することが可能です (以下参照)。

styleObj には、スタイルシートの設定 (例: フォント、テキスト装飾、文字揃え、境界線、など) を指定します。スタイルプロパティの完全な一覧については、[スタイルオブジェクトプロパティ](#) を参照ください。

任意の scope 引数を使用することで、スタイルシートをどこに定義するかを指定することができます。シートインデックス (0 起点) か、以下の定数のいずれかを渡すことができます:

- vk current sheet
- vk workbook

同じ styleName のスタイルシートが、ワークブックレベルとシートレベルとで定義されている場合、シートレベルのスタイルが優先されます。

スタイルシートを適用するには、[VP SET DEFAULT STYLE](#) または [VP SET CELL STYLE](#) コマンドを使用します。

例題

以下のコードは:

```
$styles:=New object
$styles.backColor:="green"

// 境界線オブジェクト
$borders:=New object("color";"green";"style";vk line style medium dash dot)

$styles.borderBottom:=$borders
$styles.borderLeft:=$borders
$styles.borderRight:=$borders
$styles.borderTop:=$borders

VP ADD STYLESHEET("ViewProArea";"GreenDashDotStyle";$styles)

// スタイルを適用します
VP SET CELL STYLE(VP Cells("ViewProArea";1;1;2;2);New object("name";"GreenDashDotStyle"))
```

GreenDashDotStyle という名前の、以下のようなスタイルオブジェクトを作成します:

```

{
  backColor:green,
  borderBottom:{color:green,style:10},
  borderLeft:{color:green,style:10},
  borderRight:{color:green,style:10},
  borderTop:{color:green,style:10}
}

```

参照

[4D View Pro Style Objects and Style Sheets](#)

[VP Get stylesheet](#)

[VP Get stylesheets](#)

[VP REMOVE STYLESHEET](#)

[VP SET CELL STYLE](#)

[VP SET DEFAULT STYLE](#)

VP All

`VP All (vpAreaName : Text { ; sheet : Integer }) : Object`

引数	タイプ		説明
vpAreaName	Text	->	4D View Pro フォームオブジェクト名
sheet	Integer	->	シートのインデックス (省略した場合はカレントシート)
戻り値	Object	<-	すべてのセルのレンジオブジェクト

説明

`VP All` コマンドは、すべてのセルを参照する新しいレンジオブジェクトを返します。

`vpAreaName` には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

任意の `sheet` 引数として、シートのインデックス (0 起点) を渡すことで、定義されるレンジが属するスプレッドシートを指定することができます。省略された場合、または `vk current sheet` を渡した場合、カレントスプレッドシートが使用されます。

例題

カレントスプレッドシートにおいて、全セルのレンジオブジェクトを定義します:

```
$all:=VP All("ViewProArea") // カレントシートの全セル
```

参照

[VP Cell](#)

[VP Cells](#)

[VP Column](#)

[VP Combine ranges](#)

[VP Name](#)

[VP Row](#)

C

VP Cell

`VP Cell (vpAreaName ; column : Integer ; row : Integer ; Text { ; sheet : Integer }) : Object`

引数	タイプ		説明
vpAreaName	Text	->	4D View Pro フォームオブジェクト名
column	Longint	->	シートのインデックス (省略した場合はカレントシート)
row	Longint	->	シートのインデックス (省略した場合はカレントシート)
sheet	Integer	->	シートのインデックス (省略した場合はカレントシート)
戻り値	Object	<-	すべてのセルのレンジオブジェクト

説明

VP Cell コマンドは、特定のセルを参照する新しいレンジオブジェクトを返します。

このコマンドは単一セルのレンジを想定しています。複数セルに対するレンジオブジェクトを作成するには、**VP Cells** コマンドを使用します。

vpAreaName 引数には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

column 引数で、セルレンジの位置のカラムを定義します。この引数としてカラムのインデックスを渡します。

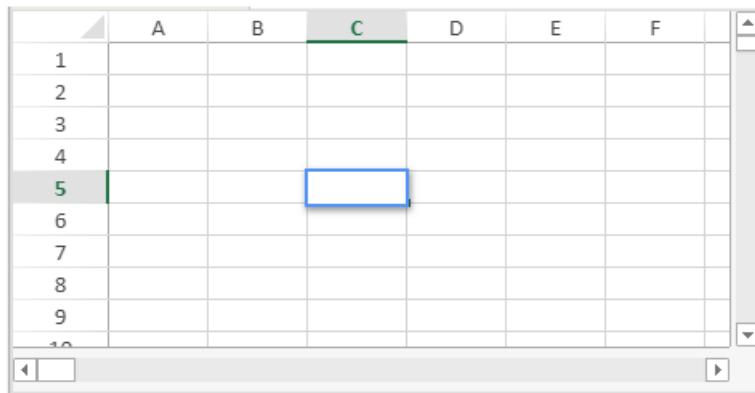
row 引数で、セルレンジの位置の行を定義します。この引数として行のインデックスを渡します。

任意の *sheet* 引数で、レンジが定義されるシートのインデックスを指定することができます。省略された場合、または `vk current sheet` を渡した場合、カレントスプレッドシートが使用されます。

インデックスは 0 起点です。

例題

以下に表示されている (カレントスプレッドシートの) セルに対するレンジオブジェクトを定義します:



以下のようにコードを書くことができます:

```
$cell:=VP Cell("ViewProArea";2;4) // C5
```

参照

[VP All](#)

[VP Cells](#)

[VP Column](#)

[VP Combine ranges](#)

[VP Name](#)

[VP Row](#)

VP Cells

`VP Cells (vpAreaName : Text ; column: Integer ; row: Integer ; columnCount : Integer ; rowCount : Integer { ; sheet : Integer }) : Object`

▶ 履歴

引数	タイプ		説明
vpAreaName	Text	->	4D View Pro フォームオブジェクト名
column	Integer	->	カラムのインデックス
row	Integer	->	行のインデックス
columnCount	Integer	->	カラム数
rowCount	Integer	->	行数
sheet	Integer	->	シートのインデックス (省略した場合はカレントシート)
戻り値	Object	<-	複数セルのレンジオブジェクト

説明

`VP Cells` コマンドは、指定された複数のセルを参照する新しいレンジオブジェクトを返します。

`vpAreaName` には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

`column` 引数で、セルレンジの先頭カラムを指定します。この引数としてカラムのインデックス (0 起点) を渡します。レンジが複数カラムにわたる場合、`columnCount` 引数も併せて使用します。

`row` 引数で、セルレンジの位置を決める行を指定します。この引数として行のインデックス (0 起点) を渡します。レンジが複数行にわたる場合、`rowCount` 引数も併せて使用します。

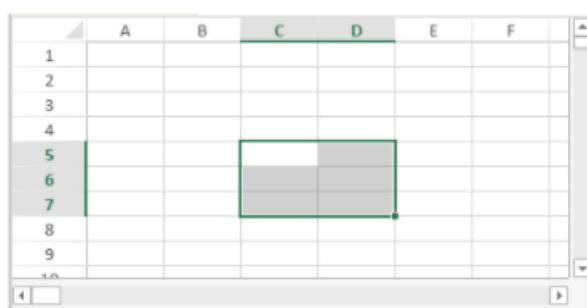
`columnCount` には、レンジに含まれるカラム数を指定することができます。`columnCount` 引数は 0 より大きい値でなければなりません。

`rowCount` には、レンジに含まれる行数を指定することができます。`rowCount` 引数は 0 より大きい値でなければなりません。

任意の `sheet` 引数として、シートのインデックス (0 起点) を渡すことで、定義されるレンジが属するスプレッドシートを指定することができます。省略された場合、または `vk current sheet` を渡した場合、カレントスプレッドシートが使用されます。

例題

(カレントシートにおいて) 以下のセルのレンジオブジェクトを定義します:



以下のようにコードを書くことができます:

```
$cells:=VP Cells("ViewProArea";2;4;2;3) // C5 から D7
```

参照

[VP All](#)

[VP Cells](#)

[VP Column](#)

[VP Combine ranges](#)

[VP Name](#)

[VP Row](#)

VP Column

VP Column (*vpAreaName* : Text ; *column*: Integer ; *columnCount* : Integer { ; *sheet* : Integer }) : Object

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro フォームオブジェクト名
<i>column</i>	Integer	->	カラムのインデックス
<i>columnCount</i>	Integer	->	カラム数
<i>sheet</i>	Integer	->	シートのインデックス (省略した場合はカレントシート)
戻り値	Object	<-	複数セルのレンジオブジェクト

説明

VP Column コマンドは、特定のカラム、あるいは複数のカラムを参照する新しいレンジオブジェクトを返します。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

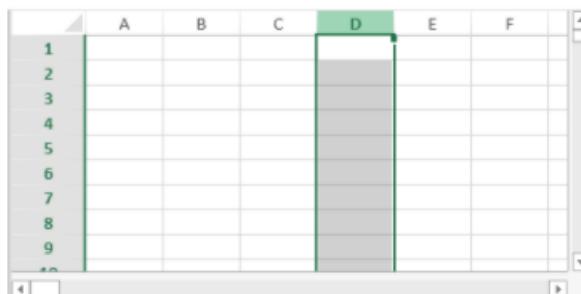
column 引数で、カラムレンジの先頭カラムを指定します。この引数としてカラムのインデックス (0 起点) を渡します。レンジが複数カラムにわたる場合には、任意の *columnCount* 引数も併せて使用します。

任意の *columnCount* には、レンジに含まれるカラム数を指定することができます。*columnCount* 引数は 0 より大きい値でなければなりません。省略時、デフォルトで値は 1 に設定され、カラム型のレンジが作成されます。

任意の *sheet* 引数として、シートのインデックス (0 起点) を渡すことで、定義されるレンジが属するスプレッドシートを指定することができます。省略された場合、または `vk current sheet` を渡した場合、カレントスプレッドシートが使用されます。

例題

以下に表示されている (カレントスプレッドシートの) カラムに対するレンジオブジェクトを定義します:



以下のようにコードを書くことができます:

```
$column:=VP Column("ViewProArea";3) // カラム D
```

参照

[VP All](#)
[VP Cells](#)
[VP Column](#)
[VP Combine ranges](#)
[VP Name](#)
[VP Row](#)
[VP SET COLUMN ATTRIBUTES](#)

VP COLUMN AUTOFIT

VP COLUMN AUTOFIT (*rangeObj* : Object)

引数	タイプ		説明
rangeObj	Object	->	レンジオブジェクト

説明

VP COLUMN AUTOFIT コマンドは、*rangeObj* 引数のレンジ内にあるカラムを、そのコンテンツに応じて自動的にリサイズします。

rangeObj 引数として、サイズを自動調整したいカラムを格納しているレンジオブジェクトを渡します。

例題

以下のカラムはすべて同じサイズで、一部のテキストが表示しきれていません:

	A	B	C	D	E
1		Hello Wor	The quick	TGIF	
2					
3					
4					
c					

カラムを選択して、以下のコードを実行すると:

```
VP COLUMN AUTOFIT(VP Get selection("ViewProarea"))
```

コンテンツに合うようにカラムがリサイズされます:

	A	B	C	D	E
1		Hello World	The quick brown fox jumped over the lazy dog.	TGIF	
2					
3					
4					
c					

参照

[VP ROW AUTOFIT](#)

VP Combine Ranges

VP Combine Ranges (*rangeObj* : Object ; *otherRangeObj* : Object {;...*otherRangeObjN* : Object}) : Object

引数	タイプ		説明
rangeObj	Object	->	レンジオブジェクト
otherRangeObj	Object	->	レンジオブジェクト
戻り値	Object	<-	統合されたレンジを格納したオブジェクト

説明

VP Combine ranges コマンドは、2つ以上のレンジオブジェクトを統合した新しいレンジオブジェクトを返します。レンジはすべて同じ 4D View Pro エリア内のものでなくてはなりません。

rangeObj には、1つ目のレンジオブジェクトを渡します。

otherRangeObj には、*rangeObj* のレンジオブジェクトと統合させる他のレンジオブジェクトを渡します。

このコマンドは *rangeObj* および *otherRangeObj* のオブジェクトを参照によって組み合わせます。

例題

セル、カラム、行のレンジオブジェクトを、新規レンジオブジェクトにまとめます:

```

$cell:=VP Cell("ViewProArea";2;4) // C5
$column:=VP Column("ViewProArea";3) // カラム D
$row:=VP Row("ViewProArea";9) // 行 10

$combine:=VP Combine ranges($cell;$column;$row)

```

参照

[VP All](#)
[VP Cells](#)
[VP Column](#)
[VP Combine ranges](#)
[VP Name](#)
[VP Row](#)
[VP SET COLUMN ATTRIBUTES](#)

VP Convert from 4D View

VP Convert from 4D View (*4DViewDocument* : Blob) : Object

引数	タイプ		説明
4DViewDocument	BLOB	->	4D View ドキュメント
戻り値	Object	<-	4D View Pro オブジェクト

説明

VP Convert from 4D View コマンドを使用すると、旧式の 4D View ドキュメントを 4D View Pro オブジェクトへと変換することができます。

旧式の 4D View プラグインが現環境にインストールされていなくても、このコマンドは使用可能です。

4DViewDocument には変換する 4D View ドキュメントを格納する BLOB変数やフィールドを渡します。コマンドは、4D View ドキュメントに保存されていた情報をすべて 4D View Pro 属性へと変換した 4D View Pro オブジェクトを返します。

例題

BLOB に保存されている 4D View エリアから 4D View Pro オブジェクトを取得します:

```

C_OBJECT($vpObj)
$vpObj:=VP Convert from 4D View($pvblob)

```

VP Convert to picture

VP Convert to picture (*vpObject* : Object {; *rangeObj* : Object}) : Picture

引数	タイプ		説明
vpObject	Object	->	変換するエリアを格納した 4D View Pro オブジェクト
rangeObj	Object	->	レンジオブジェクト
戻り値	Object	<-	エリアの SVGピクチャー

説明

VP Convert to picture コマンドは、*vpObject* 引数で指定した 4D View Pro オブジェクト (あるいは *vpObject* 内にある、*rangeObj* 引数で指定したレンジ) を、SVGピクチャーに変換します。

このコマンドは以下のような場合に有用です:

- 4D View Pro ドキュメントを 4D Write Pro ドキュメントなど、他のドキュメントに埋め込みたい場合
- 4D View Pro ドキュメントを、4D View Pro エリアに読み込まずに印刷したい場合

`vpObject` 引数には、変換したい 4D View Pro オブジェクトを渡します。このオブジェクトは事前に [VP Export to object](#) コマンドで解析するか、または [VP EXPORT DOCUMENT](#) コマンドにより保存してある必要があります。

4D View Pro エリアに含まれている式や書式 ([セルフォーマット](#) 参照) が正常に書き出されるよう、少なくともそれらが一度は評価されていることが SVG 変換プロセスには必要です。事前に評価されていないドキュメントを変換した場合、式や書式が予期せぬ形にレンダリングされる可能性があります。

`rangeObj` には、変換するセルのレンジを渡します。この引数が省略された場合のデフォルトでは、ドキュメントのコンテンツ全体が変換されます。

書式 (上の注記参照)、ヘッダーの表示状態、カラムと行などを含めた表示属性に準じて、ドキュメントコンテンツは変換されます。以下の要素の変換がサポートされます:

- テキスト: スタイル / フォント / サイズ / 文字揃え / 向き / 回転 / 書式
- セルの背景: カラー / 画像
- セルの罫線: 太さ / カラー / スタイル
- セルの結合
- ピクチャー
- 行高さ
- カラム幅
- 非表示のカラム / 行

枠線の表示状態は [VP SET PRINT INFO](#) で定義されたドキュメント属性に依存します。

ファンクションの戻り値

コマンドは SVG フォーマットのピクチャーを返します。

例題

4D View Pro エリアを SVG に変換し、結果をプレビューするためピクチャー変数に戻り値を代入します:

```
C_OBJECT($vpAreaObj)
C_PICTURE($vPict)
$vpAreaObj:=VP Export to object("ViewProArea")
$vPict:=VP Convert to picture($vpAreaObj) // エリア全体を書き出します
```

参照

[VP EXPORT DOCUMENT](#)
[VP Export to object](#)
[VP SET PRINT INFO](#)

VP Copy to object

▶履歴

`VP Copy to object (rangeObj : Object {; options : Object}) : Object`

引数	タイプ		説明
rangeObj	Object	->	レンジオブジェクト
options	Object	->	追加のオプション
戻り値	Object	<-	返されるオブジェクト。コピーされたデータが格納されています。

説明

`VP Copy to object` コマンドは、*rangeObj* のコンテンツ、スタイル、フォーミュラをオブジェクトにコピーします。

rangeObj には、コピーしたい値、フォーマット、フォーミュラを格納しているセルレンジを渡します。 *rangeObj* が結合レンジの場合は、最初のものだけが使用されます。

任意の *options* 引数として、以下のプロパティを渡すことができます。

プロパティ	タイプ	説明														
copy	Boolean	コマンド実行後もコピーされた値、書式、数式が保持するには <i>true</i> (デフォルト)。削除するには <i>false</i> 。														
copyOptions	Longint	コピーまたは移動する内容を指定します。とりうる値: <table border="1"><thead><tr><th>値</th><th>説明</th></tr></thead><tbody><tr><td><code>vk clipboard options all</code> (デフォルト)</td><td>値、フォーマット、フォーミュラを含むすべてのデータオブジェクトをコピーします。</td></tr><tr><td><code>vk clipboard options formatting</code></td><td>フォーマットだけをコピーします。</td></tr><tr><td><code>vk clipboard options formulas</code></td><td>フォーミュラだけをコピーします。</td></tr><tr><td><code>vk clipboard options formulas and formatting</code></td><td>フォーミュラとフォーマットをコピーします。</td></tr><tr><td><code>vk clipboard options values</code></td><td>値だけをコピーします。</td></tr><tr><td><code>vk clipboard options value and formatting</code></td><td>値とフォーマットをコピーします。</td></tr></tbody></table>	値	説明	<code>vk clipboard options all</code> (デフォルト)	値、フォーマット、フォーミュラを含むすべてのデータオブジェクトをコピーします。	<code>vk clipboard options formatting</code>	フォーマットだけをコピーします。	<code>vk clipboard options formulas</code>	フォーミュラだけをコピーします。	<code>vk clipboard options formulas and formatting</code>	フォーミュラとフォーマットをコピーします。	<code>vk clipboard options values</code>	値だけをコピーします。	<code>vk clipboard options value and formatting</code>	値とフォーマットをコピーします。
値	説明															
<code>vk clipboard options all</code> (デフォルト)	値、フォーマット、フォーミュラを含むすべてのデータオブジェクトをコピーします。															
<code>vk clipboard options formatting</code>	フォーマットだけをコピーします。															
<code>vk clipboard options formulas</code>	フォーミュラだけをコピーします。															
<code>vk clipboard options formulas and formatting</code>	フォーミュラとフォーマットをコピーします。															
<code>vk clipboard options values</code>	値だけをコピーします。															
<code>vk clipboard options value and formatting</code>	値とフォーマットをコピーします。															

[ワークブックオプション](#) で定義されている貼り付けオプションが考慮されます。

このコマンドは、コピーされたデータを含むオブジェクトを返します。

例題

あるレンジのコンテンツ、値、フォーマット、フォーミュラをオブジェクトに格納し、それを別のレンジに貼り付けます:

```
var $originRange; $targetRange; $dataObject; $options : Object  
  
$originRange:=VP Cells("ViewProArea"; 0; 0; 2; 5)  
  
$options:=New object  
$options.copy:=True  
$options.copyOptions:=vk clipboard options all  
  
$dataObject:=VP Copy to object($originRange; $options)  
  
$targetRange:=VP Cell("ViewProArea"; 4; 0)  
VP PASTE FROM OBJECT($targetRange; $dataObject; vk clipboard options all)
```

参照

[VP PASTE FROM OBJECT](#)

[VP MOVE CELLS](#)

[VP Get workbook options](#)

[VP SET WORKBOOK OPTIONS](#)

D

[VP DELETE COLUMNS](#)

VP DELETE COLUMNS (*rangeObj* : Object)

引数	タイプ		説明
<i>rangeObj</i>	Object	->	レンジオブジェクト

説明

VP DELETE COLUMNS コマンドは、*rangeObj* 引数で指定したカラムを削除します。

rangeObj 引数には、削除するカラムを指定するレンジオブジェクトを渡します。渡すレンジについては、以下の点に注意してください：

- レンジにカラムと行の両方が含まれる場合、カラムのみが削除されます。
- レンジに行しか含まれていない場合、コマンドは何もしません。 > カラムは右から左に向かって削除されます。

インデックスは 0 起点です。

例題

ユーザーが選択したカラムを削除します (以下の画像の B、C、D のカラムを削除します)：

A screenshot of a spreadsheet application showing a 4x9 grid. The rows are labeled 1 to 4 and the columns are labeled A to I. The range B:D is highlighted with a green background. The data in the grid is: Row 1: The, quick, brown, fox, jumped, over, the, lazy, dog. Row 2: a, b, c, d, e, f, g, h, i. Row 3: ., ., ., ., ., ., ., ., . Row 4: ., ., ., ., ., ., ., ., .

以下のコードを実行します：

```
VP DELETE COLUMNS(VP Get selection("ViewProArea"))
```

参照

[VP All](#)

[VP Cells](#)

[VP Column](#)

VP DELETE ROWS

VP DELETE ROWS (*rangeObj* : Object)

引数	タイプ		説明
<i>rangeObj</i>	Object	->	レンジオブジェクト

説明

VP DELETE ROWS コマンドは、*rangeObj* 引数で指定した行を削除します。

rangeObj 引数には、削除する行を指定するレンジオブジェクトを渡します。渡すレンジについては、以下の点に注意してください：

- レンジにカラムと行の両方が含まれる場合、行のみが削除されます。
- レンジにカラムしか含まれていない場合、コマンドは何もしません。 > 行は下から上に向かって削除されます。

インデックスは 0 起点です。

例題

ユーザーが選択した行を削除します (以下の画像の 1、2、3行目を削除します)：

	A	B	C	D	E	F	G	H	I
1									
2									
3									

The quick brown fox jumped over the lazy dog

以下のコードを実行します:

```
VP DELETE ROWS(VP Get selection("ViewProArea"))
```

参照

[VP All](#)

[VP Cells](#)

[VP Column](#)

E

VP EXPORT DOCUMENT

VP EXPORT DOCUMENT (*vpAreaName* : Text ; *filePath* : Text {; *paramObj* : Object})

引数	タイプ	説明
<i>vpAreaName</i>	Text	-> 4D View Pro フォームオブジェクト名
<i>filePath</i>	Text	-> ドキュメントのパス名
<i>paramObj</i>	Object	-> 書き出しのオプション

説明

VP EXPORT DOCUMENT コマンドは、*vpAreaName* で指定した 4D View Pro エリアに関連付けられている 4D View Pro オブジェクトを、*filePath* と *paramObj* で指定したとおりにディスク上のドキュメントに書き出します。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

filePath には書き出すドキュメントのパスとファイル名を渡します。パスを指定しない場合、ドキュメントは Project フォルダーと同階層に保存されます。

ドキュメント名に拡張子を付けることで、書き出すドキュメントの形式を指定することができます:

- 4D View Pro ("4vp")
- Microsoft Excel ("xlsx")
- PDF ("pdf")
- CSV ("txt" または "csv")

ファイル名に拡張子が含まれていなくても、*paramObj* 引数によって形式が指定されている場合、書き出されたファイルには形式に対応する拡張子がつけられます。ただし CSV 形式の場合には拡張子がつきません。

任意の *paramObj* 引数を渡すと、書き出される 4D View Pro オブジェクトの複数のプロパティに加えて、書き出しが完了した際に呼び出されるコールバックメソッド名を定義することができます。

プロパティ	タイプ	説明																		
format	text	<p>(任意) 渡した場合、書き出されるファイルの形式を指定します: ".4VP" (デフォルト)、".csv"、".xlsx"、または ".pdf"。次の定数が利用できます:</p> <ul style="list-style-type: none"> • <code>vk 4D View Pro format</code> • <code>vk csv format</code> • <code>vk MS Excel format</code> • <code>vk pdf format</code> <p>4D は必要に応じて適切な拡張子をファイル名に追加します。指定した形式が <code>filePath</code>引数として渡された拡張子と合致しない場合、指定形式の拡張子は <code>filePath</code>引数の後ろに追加されます。形式が指定されず、<code>filePath</code>引数にも拡張子がなかった場合には、デフォルトのファイル形式が使用されます。</p>																		
password	text	Microsoft Excel のみ (任意) - MS Excel ドキュメントの保護に使用されるパスワード。																		
formula	object	書き出しが完了した際に呼び出されるコールバックメソッド名。書き出しが非同期でおこなわれる (PDF および Excel 形式での書き出しが該当します) 場合かつ、書き出し後にコードを実行したい場合には、コールバックメソッドが必要です。コールバックメソッドは <code>Formula</code> コマンドと使用する必要があります (詳細は以下を参照ください)。																		
valuesOnly	boolean	フォーミュラ (あれば) の値のみを書き出すかどうかを指定します。																		
includeFormatInfo	boolean	フォーマット (書式) 情報を含めるには <code>true</code> 、それ以外の場合には <code>false</code> (デフォルトは <code>true</code>)。フォーマット情報は特定の場合 (例: SVGへの書き出しなど) において有用です。一方で、このプロパティを <code>false</code> に設定することで書き出し時間を短縮することもできます。																		
sheetIndex	number	PDF のみ (任意) - 書き出すシートのインデックス (0 起点)。-2 = 表示されている全シート (デフォルト)、-1 = カレントシートのみ																		
pdfOptions	object	<p>PDFのみ (任意) - pdf 書き出しのオプション</p> <table border="1"> <thead> <tr> <th>プロパティ</th><th>タイプ</th><th>説明</th></tr> </thead> <tbody> <tr> <td>creator</td><td>text</td><td>変換されたドキュメントの変換元を作成したアプリケーション名。</td></tr> <tr> <td>title</td><td>text</td><td>ドキュメント名。</td></tr> <tr> <td>author</td><td>text</td><td>ドキュメントの作成者の名前。</td></tr> <tr> <td>keywords</td><td>text</td><td>ドキュメントに割り当てられたキーワード。</td></tr> <tr> <td>subject</td><td>text</td><td>ドキュメントの題名。</td></tr> </tbody> </table>	プロパティ	タイプ	説明	creator	text	変換されたドキュメントの変換元を作成したアプリケーション名。	title	text	ドキュメント名。	author	text	ドキュメントの作成者の名前。	keywords	text	ドキュメントに割り当てられたキーワード。	subject	text	ドキュメントの題名。
プロパティ	タイプ	説明																		
creator	text	変換されたドキュメントの変換元を作成したアプリケーション名。																		
title	text	ドキュメント名。																		
author	text	ドキュメントの作成者の名前。																		
keywords	text	ドキュメントに割り当てられたキーワード。																		
subject	text	ドキュメントの題名。																		
csvOptions	object	<p>CSVのみ (任意) - csv 書き出しのオプション</p> <table border="1"> <thead> <tr> <th>プロパティ</th><th>タイプ</th><th>説明</th></tr> </thead> <tbody> <tr> <td>range</td><td>object</td><td>複数セルのレンジオブジェクト</td></tr> <tr> <td>rowDelimiter</td><td>text</td><td>行の区切り文字。デフォルト: "¥r¥n"</td></tr> <tr> <td>columnDelimiter</td><td>text</td><td>カラムの区切り文字。デフォルト: ","</td></tr> </tbody> </table>	プロパティ	タイプ	説明	range	object	複数セルのレンジオブジェクト	rowDelimiter	text	行の区切り文字。デフォルト: "¥r¥n"	columnDelimiter	text	カラムの区切り文字。デフォルト: ","						
プロパティ	タイプ	説明																		
range	object	複数セルのレンジオブジェクト																		
rowDelimiter	text	行の区切り文字。デフォルト: "¥r¥n"																		
columnDelimiter	text	カラムの区切り文字。デフォルト: ","																		
<customProperty>	any	コールバックメソッドの \$3 引数を通して利用可能な任意のプロパティ。																		

Excel 形式についての注意:

- 4D View Pro ドキュメントを Microsoft Excel 形式のファイルに書き出す場合、一部の設定が失われる可能性があります。たとえば、4Dメソッドとフォーミュラは Excel ではサポートされません。GrapeCity にある一覧 にて、その他の設定を確認することができます。
- このフォーマットへの書き出しが非同期に実行されるため、書き出し後にコードを実行するには、`paramObj` 引数の `formula` プロパティを使用します。

PDF 形式についての注意:

- 4D View Pro ドキュメントを PDF 形式に書き出す場合、ドキュメントで使用されているフォントは自動的に PDF ファイルに埋め込まれます。ただし、埋め込み可能なのは Unicode マップを持つ OpenType フォント (.OTF または .TTF ファイル) のみです。フォントに対して有効なフォントファイルが見つからない場合、デフォルトのフォントが代用されます。

- このフォーマットへの書き出しは非同期に実行されるため、書き出し後にコードを実行するには、*paramObj* 引数の *formula* プロパティを使用します。

CSV 形式についての注意：

- 4D View Pro ドキュメントを CSV 形式に書き出す場合、テキストと値のみが保存されるため、一部の設定が失われる可能性があります。
- すべての値は二重引用符で括られた形で保存されます。ユーザー定義区切りの値 (DSV) に関する詳細については、こちらの [Wikipedia の記事 \(英文\)](#) を参照ください。

書き出し操作が完了すると、`VP EXPORT DOCUMENT` は自動的に、*paramObj* オブジェクトの *formula* プロパティに設定されたメソッドをトリガーします (設定されていれば)。

コールバックメソッド (フォーミュラ) の渡し方

`VP EXPORT DOCUMENT` コマンドに任意の *paramObj* 引数を渡す場合、`Formula` コマンドを使って、書き出し完了時に実行される 4Dメソッドを呼び出すことができます。コールバックメソッドは、以下の値をローカル変数として受け取ります:

変数		タイプ	説明
\$1		text	4D View Pro オブジェクト名
\$2		text	書き出された 4D View Pro オブジェクトのファイルパス
\$3		object	コマンドの <i>paramObj</i> 引数への参照
\$4		object	メソッドから返されるステータスマッセージを格納したオブジェクト
.success		boolean	書き出しに成功した場合は <code>true</code> 、それ以外の場合は <code>false</code>
.errorCode		integer	エラーコード。4D あるいは JavaScript から返されます。
.errorMessage		text	エラーメッセージ。4D あるいは JavaScript から返されます。

例題 1

"VPArea" エリアのコンテンツをディスク上の 4D View Pro ドキュメントに書き出します:

```
var $docPath: Text
$docPath:="C:\\\\Bases\\\\ViewProDocs\\\\MyExport.4VP"
VP EXPORT DOCUMENT("VPArea";$docPath)
// MyExport.4VP がディスク上に保存されます
```

例題 2

カレントシートを PDF に書き出します:

```
var $params: Object
$params:=New object
$params.format:=vk pdf format
$params.sheetIndex:=-1
$params.pdfOptions:=New object("title";"Annual Report";"author";Current user)
VP EXPORT DOCUMENT("VPArea";"report.pdf";$params)
```

例題 3

4D View Pro ドキュメントを ".xlsx" 形式に書き出して、書き出し完了後にそのドキュメントをMicrosoft Excel で開くメソッドを呼び出します:

```

$params:=New object
$params.formula:=Formula(AfterExport)
$params.format:=vp MS Excel format // ".xlsx"
$params.valuesOnly:=True

VP EXPORT DOCUMENT("ViewProArea";"c:\\tmp\\convertedfile";$params)

```

AfterExport メソッド:

```

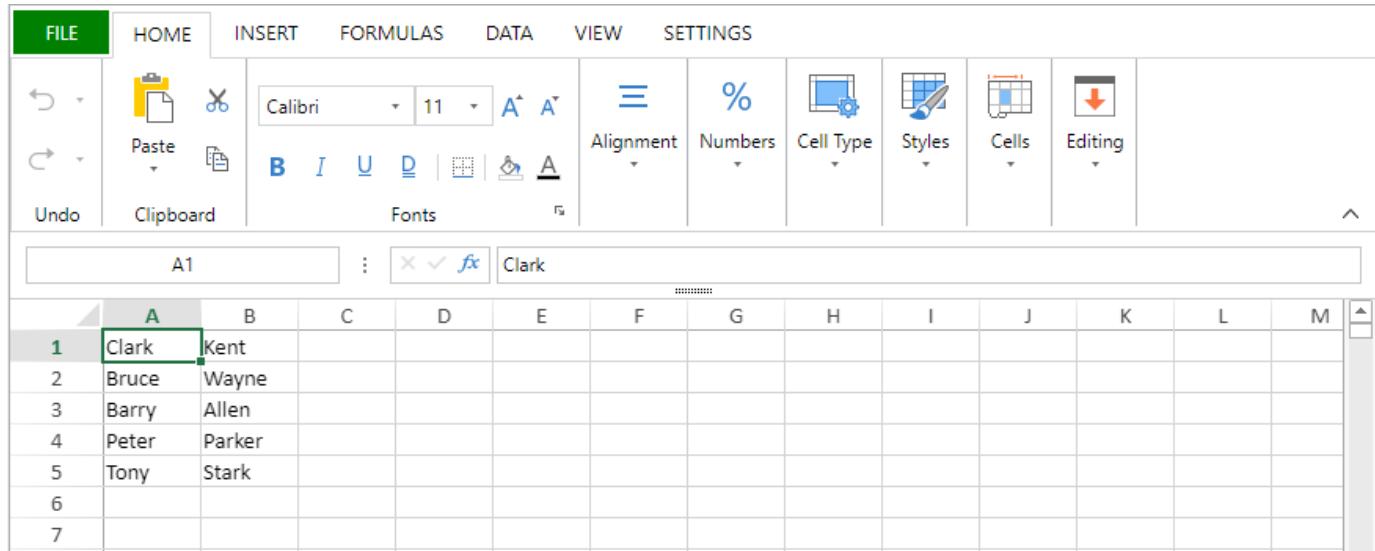
C_TEXT($1;$2)
C_OBJECT($3;$4)
$areaName:=$1
$filePath:=$2
$params:=$3
$status:=$4

If($status.success=False)
    ALERT($status.errorMessage)
Else
    LAUNCH EXTERNAL PROCESS("C:\\Program Files\\Microsoft Office\\Office15\\excel "+$filePath)
End if

```

例題 4

カレントシートを、縦棒 (|) 区切りの .txt ファイルに書き出します:



The screenshot shows a Microsoft Excel spreadsheet with the following data in columns A and B:

	A	B
1	Clark	Kent
2	Bruce	Wayne
3	Barry	Allen
4	Peter	Parker
5	Tony	Stark
6		
7		

```

var $params : Object
$params:=New object
$params.range:=VP Cells("ViewProArea";0;0;2;5)
$params.rowDelimiter:="\n"
$params.columnDelimiter:="|"
VP EXPORT DOCUMENT("ViewProArea";"c:\\tmp\\data.txt";New object("format";vk csv format;"csvOptions";$par

```

このようになります:

```

"Clark"|"Kent"
"Bruce"|"Wayne"
"Barry"|"Allen"
"Peter"|"Parker"
"Tony"|"Stark"

```

参照

[VP Convert to picture](#)

[VP Export to object](#)

[VP Column](#)

[VP Print](#)

VP Export to object

VP Export to object (*vpAreaName* : Text {; *option* : Object}) : Object

引数	タイプ	説明
<i>vpAreaName</i>	Text	-> 4D View Pro フォームオブジェクト名
<i>option</i>	Object	-> 書き出しのオプション
戻り値	Object	<- 4D View Pro オブジェクト

説明

`VP Export to object` コマンドは、*vpAreaName* で指定した 4D View Pro エリアに関連付けられている 4D View Pro オブジェクトを返します。このコマンドによって、たとえば 4D View Pro エリアを 4Dデータベースのオブジェクトフィールドに保存することができます。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

option 引数として、必要に応じて以下の書き出しオプションを渡すことができます:

プロパティ	タイプ	説明
includeFormatInfo	boolean	フォーマット（書式）情報を含めるには true、それ以外の場合には false（デフォルトは true）。フォーマット情報は特定の場合（例：SVGへの書き出しなど）において有用です。一方で、このプロパティを false に設定することで書き出し時間を短縮することができます。

4D View Pro オブジェクトについての詳細は [4D View Pro オブジェクト](#) を参照ください。

例題 1

4D View Pro エリアの "version" プロパティを取得します:

```
var $vpAreaObj : Object
var $vpVersion : Number
$vpAreaObj:=VP Export to object("vpArea")
// $vpVersion:=OB Get($vpAreaObj;"version")
$vpVersion:=$vpAreaObj.version
```

例題 2

フォーマット（書式）情報を含めてエリアを書き出します:

```
var $vpObj : Object
$vpObj:=VP Export to object("vpArea";New object("includeFormatInfo";False))
```

参照

[VP Convert to picture](#)

[VP EXPORT DOCUMENT](#)

[VP IMPORT FROM OBJECT](#)

VP Find

VP Find (*rangeObj* : Object ; *searchValue* : Text) : Object

VP Find (*rangeObj* : Object ; *searchValue* : Text ; *searchCondition* : Object }) : Object

VP Find (*rangeObj* : Object ; *searchValue* : Text ; *searchCondition* : Object ; *replaceValue* : Text) : Object

引数	タイプ		説明
<i>rangeObj</i>	Object	->	レンジオブジェクト
<i>searchValue</i>	Text	->	検索値
<i>searchCondition</i>	Object	->	検索条件を格納したオブジェクト
<i>replaceValue</i>	Text	->	置き換え値
戻り値	Object	<-	レンジオブジェクト

説明

VP Find コマンドは、*rangeObj* に指定したレンジ内で *searchValue* に指定した値を検索します。任意の引数を渡すことで、検索条件を詳細に指定したり、検索結果を置換したりすることができます。

rangeObj 引数として、検索対象のレンジを格納したオブジェクトを渡します。

searchValue 引数として、*rangeObj* に指定したレンジ内で検索するテキスト値を渡します。

任意の *searchCondition* 引数を渡すことで、検索がどのように実行されるかを指定することができます。以下のオブジェクトプロパティがサポートされています：

プロパティ	タイプ	説明	
afterColumn	Integer	検索を開始するカラムの直前のカラムの番号。 <i>rangeObj</i> 引数が統合されたレンジの場合、渡されるカラムの番号は最初のレンジのものでなければなりません。デフォルト値: -1 (<i>rangeObj</i> の最初)	
afterRow	Integer	検索を開始する行の直前の行番号。 <i>rangeObj</i> 引数が統合されたレンジの場合、渡される行番号は最初のレンジのものでなければなりません。デフォルト値: -1 (<i>rangeObj</i> の最初)	
all	Boolean	<ul style="list-style-type: none"> • true - <i>rangeObj</i> 内で <i>searchValue</i> の値に合致するセルはすべて返されます。 • false - (デフォルト値) <i>rangeObj</i> 内で <i>searchValue</i> の値に合致する最初のセルのみが返されます。 	
flags	Integer	<code>vk find flag exact match</code>	セルの中身全体が検索値と完全に一致する必要があります
		<code>vk find flag ignore case</code>	文字の大小は区別されません。例: "a" と "A" は同じとみなされます。
		<code>vk find flag none</code>	検索フラグは指定されていません (デフォルト)。
		<code>vk find flag use wild cards</code>	<p>検索文字列においてワイルドカード文字 (*,?) を使用できます。ワイルドカードは、すべての文字列の比較に使用することができ、ワイルドカードによって置き換わる文字の数は指定されません:</p> <ul style="list-style-type: none"> • * は 0 から複数文字に使用可能です (例: "bl*" を検索した場合、"bl"、"black"、"blob" などが合致します)。 • ? は単一文字に使用可能です (例: "h?t" を検索した場合、"hot"、"hit" などが合致します)。
<p>フラグは組み合わせることができます。たとえば:</p> <pre>\$search.flags:=vk find flag use wild cards+vk find flag ignore case</pre>			
order	Integer	<code>vk find order by columns</code>	検索がカラムごとに実行されます。カラムの各行が検索されたあとに次のカラムへと移動します。
		<code>vk find order by rows</code>	検索が行ごとに実行されます。行の各カラムが検索されたあとに次の行へと移動します (デフォルト)。
target	Integer	<code>vk find target formula</code>	セルフォーミュラ内で検索がおこなわれます。
		<code>vk find target tag</code>	セルタグ内で検索がおこなわれます。
		<code>vk find target text</code>	セルテキスト内で検索がおこなわれます (デフォルト)。
<p>フラグは組み合わせることができます。たとえば:</p> <pre>\$search.target:=vk find target formula+vk find target text</pre>			

任意の *replaceValue* 引数として、*rangeObj* 内で見つかった *searchValue* の値のテキストを置換するテキストを渡すことができます。

返されるオブジェクト

この関数は、検出または置換された検索値の詳細を格納したレンジオブジェクトを返します。何も見つからなかった場合には、空のレンジオブジェクトが返されます。

例題 1

"Total" という単語が入っている最初のセルを見つけるには:

```

var $range;$result : Object

$range:=VP All("ViewProArea")

$result:=VP Find($range;"Total")

```

例題 2

"Total" のセルを検出し、それを "Grand Total" で置き換えるには:

```

var $range;$condition;$result : Object

$range:=VP All("ViewProArea")

$condition:=New object
$condition.target:=vk find target text
$condition.all:=True // ドキュメント全体を検索します
$condition.flags:=vk find flag exact match

// カレントシートにおいて "Total" のみを格納しているセルを "Grand Total" で置き換えます
$result:=VP Find($range;"Total";$condition;"Grand Total")

// 戻り値のレンジオブジェクトが空かどうかをチェックします
If($result.ranges.length=0)
    ALERT("No result found")
Else
    ALERT($result.ranges.length+" results found")
End if

```

VP FLUSH COMMANDS

VP FLUSH COMMANDS (*vpAreaName* : Text)

引数	タイプ	説明
<i>vpAreaName</i>	Text	-> 4D View Pro フォームオブジェクト名

説明

VP FLUSH COMMANDS コマンドは、保存されているコマンドをただちに実行し、コマンドバッファをクリアします。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

パフォーマンス向上と、送信リクエスト数を抑えるため、デベロッパーが呼び出した 4D View Pro コマンドはコマンドバッファに保存されます。 VP FLUSH COMMANDS は呼び出されると、メソッド終了時にコマンドをバッチとして実行し、コマンドバッファのコンテンツを空にします。

例題

コマンドの実行をトレースし、コマンドバッファを空にします:

```

VP SET TEXT VALUE(VP Cell("ViewProArea1";10;1);"INVOICE")
VP SET TEXT VALUE(VP Cell("ViewProArea1";10;2); "Invoice date: ")
VP SET TEXT VALUE(VP Cell("ViewProArea1";10;3); "Due date: ")

VP FLUSH COMMANDS(("ViewProArea1")
TRACE

```

VP Font to object

VP Font to object (*font* : Text) : Object

引数	タイプ		説明
<i>font</i>	Text	->	フォントのショートハンド文字列

説明

VP Font to object ユーティリティコマンドは、フォントのショートハンド文字列からオブジェクトを返します。このオブジェクトはその後、オブジェクト記法を通してフォントプロパティ設定を取得・設定するのに使用することができます。

font には、フォントのショートハンド文字列を渡してフォントのプロパティを指定します（例: "12 pt Arial"）。フォントのショートハンド文字列についての詳細は、[こちら](#) を参照ください。

返されるオブジェクトには、フォント属性がプロパティとして格納されています。利用可能なプロパティの詳細については、[VP Object to font](#) コマンドを参照ください。

例題 1

以下のコードを実行すると：

```
$font:=VP Font to object("16pt arial")
```

以下の \$font オブジェクトが返されます：

```
{
  family:arial
  size:16pt
}
```

例題 2

[VP Object to font](#) の例題を参照ください。

参照

[4D View Pro Style Objects and Style Sheets](#)

[VP Object to font](#)

[VP SET CELL STYLE](#)

[VP SET DEFAULT STYLE](#)

G

VP Get active cell

VP Get active cell (*vpAreaName* : Text { ; *sheet* : Integer }) : Object

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro フォームオブジェクト名
<i>sheet</i>	Integer	->	シートのインデックス（省略した場合はカレントシート）
戻り値	Object	<-	単一セルのレンジオブジェクト

説明

VP Get active cell コマンドは、フォーカスを持ち、データ入力されようとしているセル（アクティブセル）を参照する新しいレンジオブジェクトを返します。

`vpAreaName` には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

任意の `sheet` 引数として、シートのインデックス (0 起点) を渡すことで、定義されるレンジが属するスプレッドシートを指定することができます。省略された場合、または `vk current sheet` を渡した場合、カレントスプレッドシートが使用されます。

例題

A	B	C	D	E	F	G
1						
2	Month	Area	Mr. Smith	Ms. Johnson	Ms. Williams	Mr. Jones
3	January	South	1898	1857	1651	1448
4		East	4859	4857	2548	4876
5		North	2458	1524	6150	4987
6		West	5787	1580	3975	4878
7	Total		15002	9818	14324	16189
8	February	South	6668	4374	17495	9999
9		East	5955	1677	7944	9400
10		North	1000	6722	2195	2777
11		West	6896	8355	7195	2058
12	Total		20519	21128	34829	24234
13	March	South	2577	2000	6185	2704
14		East	4859	4857	2548	4876
15		North	2458	1524	6150	4987
16		West	5787	1580	3975	4878
17	Total		15621	9961	18858	17445

以下のコードを実行するとアクティブセルの座標が取得できます:

```
$activeCell:=VP Get active cell("myVPArea")

// 返されるレンジオブジェクトには以下が格納されています:
// $activeCell.ranges[0].column=3
// $activeCell.ranges[0].row=4
// $activeCell.ranges[0].sheet=0
```

参照

[VP ADD SELECTION](#)

[VP Get selection](#)

[VP RESET SELECTION](#)

[VP SET ACTIVE CELL](#)

[VP SET SELECTION](#)

[VP SHOW CELL](#)

VP Get cell style

`VP Get cell style (rangeObj : Object) : Object`

引数	タイプ		説明
rangeObj	Object	->	レンジオブジェクト
戻り値	Object	<-	スタイルオブジェクト

説明

`VP Get cell style` コマンドは、`rangeObj` 引数で指定したレンジの最初のセルの [スタイルオブジェクト](#) を返します。

`rangeObj` 引数で、スタイルを取得するレンジを指定します。

- `rangeObj` 引数としてセルレンジを渡した場合、セルのスタイルが返されます。
- `rangeObj` 引数として、セルレンジではないレンジを渡した場合、そのレンジ内の最初のセルのスタイルが返されます。
- `rangeObj` 引数に複数のレンジが含まれている場合、最初のレンジの最初のセルのスタイルのみが返されます。

例題

選択されたセル (B2) のスタイルの詳細を取得します:

	A	B	C
1			
2		H e l l o	
3		W o r l d	

以下のコードを実行すると:

```
$cellStyle:=VP Get cell style(VP Get selection("myDoc"))
```

... 以下のオブジェクトが返されます:

```
{
  "backColor":"Azure",
  "borderBottom": {
    "color": "#800080",
    "style": 5
  },
  "font": "8pt Arial",
  "foreColor": "red",
  "hAlign": 1,
  "isVerticalText": "true",
  "vAlign": 0
}
```

参照

[VP GET DEFAULT STYLE](#)

[VP SET CELL STYLE](#)

VP Get column attributes

VP Get column attributes (*rangeObj* : Object) : Collection

引数	タイプ		説明
<i>rangeObj</i>	Object	->	レンジオブジェクト
戻り値	Collection	<-	カラムプロパティのコレクション

説明

VP Get column attributes コマンドは、*rangeObj* 引数で指定したレンジ内にあるカラムのプロパティのコレクションを返します。

rangeObj 引数には、属性を取得したいカラムのレンジを格納しているオブジェクトを渡します。

[VP SET COLUMN ATTRIBUTES](#) コマンドを使用して属性を設定する/いないに関わらず、返されるコレクションにはカラムの属性がすべて返されます。

例題

以下のコードは:

```
C_OBJECT($range)
C_COLLECTION($attr)

$range:=VP Column("ViewProArea";1;2)
$attr:=VP Get column attributes($range)
```

渡したレンジ内の属性のコレクションを返します:

length	2
\$attr[0]	{"width":150,"pageBreak":false,"visible":true,"resizable":false,"header":"Hello World"}
\$attr[0].header	"Hello World"
\$attr[0].pageBreak	False
\$attr[0].resizable	False
\$attr[0].visible	True
\$attr[0].width	150
\$attr[1]	{"width":62,"pageBreak":false,"visible":true,"resizable":true,"header":"C"}
\$attr[1].header	"C"
\$attr[1].pageBreak	False
\$attr[1].resizable	True
\$attr[1].visible	True
\$attr[1].width	62

参照

[VP Get row attributes](#)

[VP SET COLUMN ATTRIBUTES](#)

[VP SET ROW ATTRIBUTES](#)

VP Get column count

VP Get column count (vpAreaName : Text { ; sheet : Integer }) : Integer

引数	タイプ		説明
vpAreaName	Text	->	4D View Pro エリアフォームオブジェクト名
sheet	Integer	->	シートのインデックス (省略した場合はカレントシート)
戻り値	Integer	<-	カラムの総数

説明

VP Get column count コマンドは、vpAreaName 引数で指定した 4D View Pro エリア内の、sheet 引数で指定したシートにおけるカラムの総数を返します。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

任意の sheet 引数にシートインデックス (0 起点) を指定することで、どのシートのカラム数を取得するかを定義することができます。省略された場合、または `vk current sheet` を渡した場合、カレントスプレッドシートが使用されます。

例題

以下のコードを実行すると、4D View Pro エリア内のカラムの数が返されます:

```
C_Integer($colCount)
$colCount:=VP Get column count("ViewProarea")
```

参照

[VP Get row count](#)
[VP SET COLUMN COUNT](#)
[VP SET ROW COUNT](#)

VP Get current sheet

VP Get current sheet (*vpAreaName* : Text)

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro フォームオブジェクト名
ファンクションの戻り値	Integer	<-	カレントシートのインデックス

説明

`VP Get current sheet` コマンドは、*vpAreaName* 引数で指定した View Pro エリアのカレントシートのインデックスを返します。カレントシートとは、ドキュメント内で選択されているシートのことです。

vpAreaName には、4D View Pro エリアの名前を渡します。

インデックスは 0 起点です。

例題

3番目のシートが選択されている場合:



コマンドは 2 を返します:

```
$index:=VP Get current sheet("ViewProArea")
```

参照

[VP SET CURRENT SHEET](#)

VP Get default style

VP Get default style (*vpAreaName* : Text { ; *sheet* : Integer }) : Integer

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro エリアフォームオブジェクト名
<i>sheet</i>	Integer	->	シートのインデックス (省略した場合はカレントシート)
戻り値	Integer	<-	カラムの総数

説明

`VP Get default style` コマンドは、シートのデフォルトスタイルオブジェクトを返します。返されるオブジェクトには、ドキュメントの基本的なレンダリングプロパティに加え、[VP SET DEFAULT STYLE](#) コマンドによって事前に設定されたデフォルトのスタイル設定 (あれば) が格納されます。スタイルプロパティの詳細な情報については、[スタイルオブジェクトとスタイルシート](#) を参照ください。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

任意の *sheet* 引数にシートインデックス (0 起点) を指定することで、どのシートのカラム数を取得するかを定義することができます。省略された場合、または `vk current sheet` を渡した場合、カレントスプレッドシートが使用されます。

例題

このドキュメントのデフォルトスタイルを取得したい場合:

	A	B	C
1			
2		Hello World!	
3			
4			
5			
6			
7			
8			

以下のコードを実行すると:

```
$defaultStyle:=VP Get default style("myDoc")
```

\$defaultStyle オブジェクトに以下のような情報が返されます:

```
{
    backColor:#E6E6FA,
    hAlign:0,
    vAlign:0,
    font:12pt papyrus
}
```

参照

[VP Get cell style](#)

[VP SET DEFAULT STYLE](#)

VP Get formula

VP Get formula (rangeObj : Object) : Text

引数	タイプ		説明
rangeObj	Object	->	レンジオブジェクト
戻り値	Text	<-	Formula

説明

VP Get formula コマンドは、指定したセルレンジのフォーミュラを取得します。

rangeObj 引数で、フォーミュラを取得したいレンジを指定します。rangeObj 引数のレンジが複数セルあるいは複数レンジを指定している場合、最初のセルのフォーミュラが返されます。rangeObj 引数がフォーミュラのないセルを指定している場合、コマンドは空の文字列を返します。

例題

```
// フォーミュラを設定します
VP SET FORMULA(VP Cell("ViewProArea";5;2); "SUM($A$1:$C$10)")

$result:=VP Get formula(VP Cell("ViewProArea";5;2)) // $result="SUM($A$1:$C$10)"
```

参照

[VP Get formulas](#)

[VP SET FORMULA](#)

[VP SET ROW COUNT](#)

VP Get formula by name

VP Get formula by name (*vpAreaName* : Text ; *name* : Text { ; *scope* : Number }) : Object

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro フォームオブジェクト名
<i>name</i>	Text	->	命名レンジの名前
<i>scope</i>	Number	->	ターゲットのスコープ (デフォルト=カレントシート)
戻り値	Text	<-	命名フォーミュラ、または命名レンジの定義

説明

VP Get formula by name コマンドは、*name* 引数で指定された名前の命名フォーミュラ、あるいは命名レンジに対応したフォーミュラとコメントを返します。定義されたスコープにそれらが存在しない場合には null が返されます。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

name には、取得したい命名レンジあるいは命名フォーミュラの名前を渡します。命名レンジは、絶対セル参照を格納したフォーミュラとして返されるという点に注意してください。

scope 引数を使用することで、フォーミュラを取得するスコープを定義できます。その際、シートのインデックス (0 起点) を渡すか、以下の定数のいずれかを渡します：

- `vk current sheet`
- `vk workbook`

返されるオブジェクト

戻り値のオブジェクトには、以下のプロパティが格納されています：

プロパティ	タイプ	説明
<i>formula</i>	Text	命名フォーミュラまたは命名レンジに対応したフォーミュラのテキスト。命名レンジの場合、フォーミュラは連続した絶対セル参照として返されます。
<i>comment</i>	Text	命名フォーミュラまたは命名レンジに対応したコメント

例題

```
$range:=VP Cell("ViewProArea";0;0)
VP ADD RANGE NAME("Total1";$range)

$formula:=VP Get formula by name("ViewProArea";"Total1")
//$formula.formula=Sheet1!$A$1

$formula:=VP Get formula by name("ViewProArea";"Total")
//$formula=null (存在しない場合)
```

参照

[VP ADD FORMULA NAME](#)

[VP ADD RANGE NAME](#)

[VP Get names](#)

VP Get formulas

VP Get formulas (*rangeObj* : Object) : Collection

引数	タイプ		説明
<i>rangeObj</i>	Object	->	レンジオブジェクト
戻り値	Collection	<-	フォーミュラ値のコレクション

説明

VP Get formulas コマンドは、*rangeObj* 引数で指定したレンジからフォーミュラを取得します。

rangeObj 引数で、フォーミュラを取得したいレンジを指定します。*rangeObj* 引数のレンジが複数レンジを指定している場合、最初のレンジのフォーミュラが返されます。*rangeObj* 引数のレンジにフォーミュラが一つも含まれていない場合には、コマンドは空の文字列を返します。

返されるコレクションは 2次元構造になっています：

- 第1レベルのコレクションは、フォーミュラのサブコレクションを格納しています。それぞれのサブコレクションは行をあらわします。
- それぞれのサブコレクションは行におけるセルの値を定義します。値は、セルのフォーミュラを格納しているテキスト要素です。

例題

このドキュメントの総計行と平均行のフォーミュラを取得します：

A	B	C	D	E	F	G
1		Data				
2	1	2	3	6	2	
3	4	5	6	15	5	
4	7	8	9	24	8.5	
5						
=						

以下のコードを使用することができます：

```
$formulas:=VP Get formulas(VP Cells("ViewProArea";5;1;2;3))
// $formulas[0]=[Sum(B2:D2),Average(B2:D2)]
// $formulas[1]=[Sum(B3:D3),Average(B3:D3)]
// $formulas[2]=[Sum(B4:D4),Average(C4:D4)]
```

参照

[VP Get formula](#)

[VP Get values](#)

[VP SET FORMULAS](#)

[VP SET VALUES](#)

VP Get frozen panes

VP Get frozen panes (*vpAreaName* : Text { ; *sheet* : Integer }) : Object

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro フォームオブジェクト名
<i>sheet</i>	Integer	->	シートのインデックス (省略した場合はカレントシート)
戻り値	Object	<-	固定化されたカラムと行についての情報を格納したオブジェクト

説明

VP Get frozen panes コマンドは、*vpAreaName* 引数で指定した View Pro エリア内の、固定化されたカラムと行についての情報を格納したオブジェクトを返します。

`vpAreaName` には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

任意の `sheet` 引数として、シートのインデックス (0 起点) を渡すことで、定義されるレンジが属するスプレッドシートを指定することができます。省略された場合、または `vk current sheet` を渡した場合、カレントスプレッドシートが使用されます。

返されるオブジェクト

このコマンドは固定化されたカラムと行についてのオブジェクトを返します。このオブジェクトには、次のプロパティが格納があります:

プロパティ	タイプ	説明
<code>columnCount</code>	<code>Integer</code>	シートの左側にある固定化されたカラム
<code>trailingColumnCount</code>	<code>Integer</code>	シートの右側にある固定化されたカラム
<code>rowCount</code>	<code>Integer</code>	シートの上側にある固定化された行
<code>trailingRowCount</code>	<code>Integer</code>	シートの下側にある固定化された行

例題

固定化されたカラムと行についての情報を取得します:

```
var $panesObj : Object  
$panesObj:=VP Get frozen panes("ViewProArea")
```

戻り値のオブジェクトには、以下のようなものが格納されています:

Expression	Value
✓ \$paneObj	{"columnCount":3,"trailingColumnCount":0,"rowCount":1,"trailingRowCount":0}
└ columnCount	3
└ rowCount	1
└ trailingColumnCount	0
└ trailingRowCount	0

参照

[VP SET FROZEN PANES](#)

VP Get names

`VP Get names` (`vpAreaName` : Text { ; `scope` : Number }) : Collection

引数	タイプ		説明
<code>vpAreaName</code>	Text	->	4D View Pro フォームオブジェクト名
<code>scope</code>	Number	->	ターゲットのスコープ (デフォルト = カレントシート)
戻り値	Collection	<-	定義されたスコープ内に存在する名前

説明

`VP Get names` コマンドは、カレントシートまたは `scope` 引数で指定されたスコープ内において定義されているすべての "名前" のコレクションを返します。

`vpAreaName` には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

`scope` 引数を使用することで、名前を取得するスコープを定義できます。その際、シートのインデックス (0 起点) を渡すか、以下の定数のいずれかを渡します:

- `vk current sheet`
- `vk workbook`

返されるコレクション

返されるコレクションには、1つの名前につき 1つのオブジェクトが格納されています。以下のオブジェクトプロパティが返されます:

プロパティ	タイプ	説明
<code>result[].name</code>	Text	セルまたはレンジ名
<code>result[].formula</code>	Text	formula
<code>result[].comment</code>	Text	名前に割り当てられたコメント

返されるプロパティは、命名された要素のタイプ（命名セル、命名レンジ、または命名フォーミュラ）に応じて異なります。

例題

```
var $list : Collection
$list:=VP Get names("ViewProArea";2) // 3番目のシートにある名前
```

参照

[VP ADD FORMULA NAME](#)
[VP ADD RANGE NAME](#)
[VP Get formula by name](#)
[VP Name](#)

VP Get print info

`VP Get print info (vpAreaName : Text { ; sheet : Integer }) : Object`

引数	タイプ		説明
<code>vpAreaName</code>	Text	->	4D View Pro フォームオブジェクト名
<code>sheet</code>	Integer	->	シートのインデックス（省略した場合はカレントシート）
戻り値	Object	<-	印刷情報のオブジェクト

説明

`VP Get print info` コマンドは、`vpAreaName` 引数で指定したエリアの印刷属性を格納したオブジェクトを返します。

`vpAreaName` には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

任意の `sheet` 引数として、シートのインデックス（0 起点）を渡すことで、印刷属性を取得するスプレッドシートを指定することができます。省略された場合、または `vk current sheet` を渡した場合、カレントスプレッドシートが使用されます。

例題

以下のコードを実行すると:

```
$pinfo:=VP Get print info("ViewProArea")
```

`VP SET PRINT INFO` コマンドで設定された、4D View Pro エリアの以下のような印刷属性を返します:

```
{
bestFitColumns:false,
bestFitRows:false,
blackAndWhite:false,
centering:0,
columnEnd:8,
columnStart:0,
firstPageNumber:1,
fitPagesTall:1,
fitPagesWide:1,
footerCenter:"&S.H.I.E.L.D. & Sales Per Region",
footerCenterImage:, 
footerLeft:, 
footerLeftImage:, 
footerRight:"page &P of &N", 
footerRightImage:, 
headerCenter:, 
headerCenterImage:, 
headerLeft:&G, 
headerLeftImage:logo.png, 
headerRight:, 
headerRightImage:, 
margin:{top:75,bottom:75,left:70,right:70,header:30,footer:30}, 
orientation:2, 
pageOrder:0, 
pageRange:, 
paperSize:{width:850,height:1100,kind:1}, 
qualityFactor:2, 
repeatColumnEnd:-1, 
repeatColumnStart:-1, 
repeatRowEnd:-1, 
repeatRowStart:-1, 
rowEnd:24, 
rowStart:0, 
showBorder:false, 
showColumnHeader:0, 
showGridLine:false, 
showRowHeader:0, 
useMax:true, 
watermark:[], 
zoomFactor:1
}
```

参照

[4D View Pro Print Attributes](#)
[VP SET PRINT INFO](#)

VP Get row attributes

VP Get row attributes (rangeObj : Object) : Collection

引数	タイプ		説明
rangeObj	Object	->	レンジオブジェクト
戻り値	Collection	<-	行プロパティのコレクション

説明

`VP Get row attributes` コマンドは、`rangeObj` 引数で指定したレンジ内にある行のプロパティのコレクションを返します。

`rangeObj` 引数には、属性を取得したい行のレンジを格納しているオブジェクトを渡します。

[VP SET ROW ATTRIBUTES](#) コマンドを使用して属性を設定する/いないに関わらず、返されるコレクションには行の属性がすべて返されます。

例題

以下のコードは、指定したレンジ内の行属性のコレクションを返します：

```
var $range : Object
var $attr : Collection

$range:=VP Column("ViewProArea";1;2)
$attr:=VP Get row attributes($range)

  ↗ length          2
  ↗ $attr[0]
    ↗ $attr[0].header      "june"
    ↗ $attr[0].height       75
    ↗ $attr[0].pageBreak     False
    ↗ $attr[0].resizable      True
    ↗ $attr[0].visible        True
  ↗ $attr[1]
    ↗ $attr[1].header      "3"
    ↗ $attr[1].height       20
    ↗ $attr[1].pageBreak     False
    ↗ $attr[1].resizable      True
    ↗ $attr[1].visible        True
```

参照

[VP Get column attributes](#)
[VP SET COLUMN ATTRIBUTES](#)
[VP SET ROW ATTRIBUTES](#)

VP Get row count

VP Get row count (*vpAreaName* : Text {; *sheet* : Integer }) : Integer

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro エリアフォームオブジェクト名
<i>sheet</i>	Integer	->	シートのインデックス (省略した場合はカレントシート)
戻り値	Integer	<-	行の総数

説明

`VP Get row count` コマンドは、*vpAreaName* 引数で指定した 4D View Pro エリア内、*sheet* 引数で指定したシートにおける行の総数を返します。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

任意の *sheet* 引数にシートインデックス (0 起点) を指定することで、どのシートの行数を取得するかを定義することができます。省略された場合、または `vk current sheet` を渡した場合、カレントスプレッドシートが使用されます。

例題

以下のコードを実行すると、4D View Pro エリア内の行の数が返されます：

```

var $rowCount : Integer
$rowCount:=VP Get row count("ViewProarea")

```

参照

[VP Get column count](#)
[VP SET COLUMN COUNT](#)
[VP SET ROW COUNT](#)

VP Get selection

VP Get selection (vpAreaName : Text {; sheet : Integer }) : Object

引数	タイプ		説明
vpAreaName	Text	->	4D View Pro エリアフォームオブジェクト名
sheet	Integer	->	シートのインデックス (省略した場合はカレントシート)
戻り値	Object	<-	複数セルのレンジオブジェクト

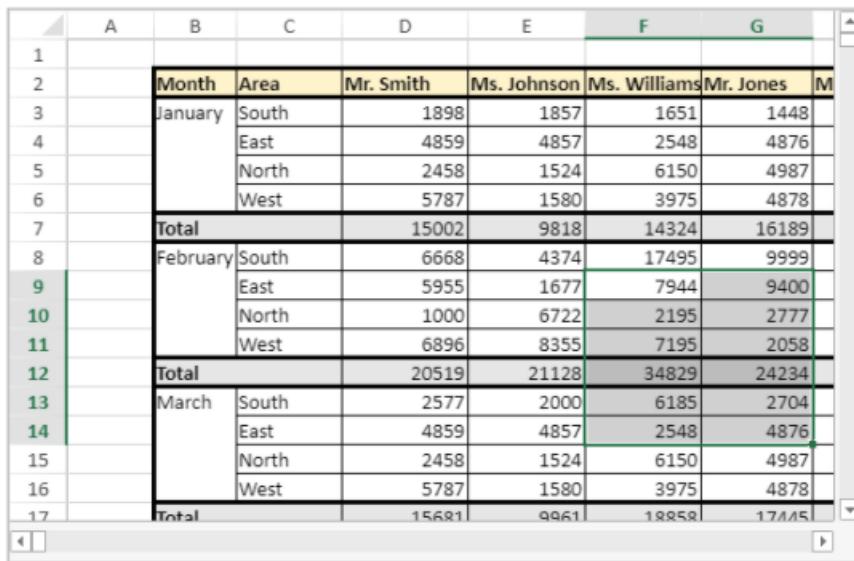
説明

`VP Get selection` コマンドは、現在選択されているセルを参照する新しいレンジオブジェクトを返します。

`vpAreaName` には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

任意の `sheet` 引数として、シートのインデックス (0 起点) を渡すことで、定義されるレンジが属するスプレッドシートを指定することができます。省略された場合、または `vk current sheet` を渡した場合、カレントスプレッドシートが使用されます。

例題



A	B	C	D	E	F	G
1						
2	Month	Area	Mr. Smith	Ms. Johnson	Ms. Williams	Mr. Jones
3	January	South	1898	1857	1651	1448
4		East	4859	4857	2548	4876
5		North	2458	1524	6150	4987
6		West	5787	1580	3975	4878
7	Total		15002	9818	14324	16189
8	February	South	6668	4374	17495	9999
9		East	5955	1677	7944	9400
10		North	1000	6722	2195	2777
11		West	6896	8355	7195	2058
12	Total		20519	21128	34829	24234
13	March	South	2577	2000	6185	2704
14		East	4859	4857	2548	4876
15		North	2458	1524	6150	4987
16		West	5787	1580	3975	4878
17	Total		15621	99611	182521	171151

以下のコードを実行すると、現在選択されているセルの座標がすべて取得できます：

```

$currentSelection:=VP Get selection("myVPArea")

//returns a range object containing:
//$currentSelection.ranges[0].column=5
//$currentSelection.ranges[0].columnCount=2
//$currentSelection.ranges[0].row=8
//$currentSelection.ranges[0].rowCount=6

```

参照

[VP ADD SELECTION](#)
[VP Get active cell](#)
[VP SET ACTIVE CELL](#)
[VP SET SELECTION](#)
[VP SHOW CELL](#)

VP Get sheet count

VP Get sheet count (*vpAreaName* : Text) : Integer

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro フォームオブジェクト名
ファンクションの戻り値	Integer	<-	シートの数

説明

VP Get sheet count コマンドは、*vpAreaName* で指定したエリアにロードされているドキュメント内にあるシート数を返します。

vpAreaName には、4D View Pro エリアの名前を渡します。

例題

以下のドキュメントにおいて:



シート数を取得し、最後のシートをカレントシートに設定します:

```
$count:=VP Get sheet count("ViewProArea")
// 最後のシートをカレントシートに設定します (0 起点)
VP SET CURRENT SHEET("ViewProArea";$count-1)
```



参照

[VP Get sheet index](#)
[VP SET SHEET COUNT](#)

VP Get sheet index

VP Get sheet index (*vpAreaName* : Text ; *name* : Text) : Integer

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro フォームオブジェクト名
<i>name</i>	Text	->	シート名
ファンクションの戻り値	Integer	<-	シートのインデックス

説明

VP Get sheet index コマンドは、*vpAreaName* 引数で指定したエリア内の、シート名で指定したシートのインデックスを返します。

vpAreaName には、4D View Pro エリアの名前を渡します。

name には、インデックスを返して欲しいシートの名前を渡します。 *name* 引数のシート名がドキュメント内に見つからない場合、コマンドは -1 を返します。

インデックスは 0 起点です。

例題

以下のドキュメントにおいて:



"Total first quarter" という名前のシートのインデックスを取得します:

```
$index:=VP Get sheet index("ViewProArea";"Total first quarter") // 2 を返します
```

参照

[VP Get sheet count](#)

[VP Get sheet name](#)

VP Get sheet name

VP Get sheet name (*vpAreaName* : Text ; *sheet* : Integer) : Text

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro フォームオブジェクト名
<i>sheet</i>	Integer	->	シートのインデックス
ファンクションの戻り値	Text	<-	シート名

説明

`VP Get sheet name` コマンドは、*vpAreaName* で指定したエリア内の、インデックスで指定したシートの名前を返します。

vpAreaName には、4D View Pro エリアの名前を渡します。

sheet には、名前を返して欲しいシートのインデックスを渡します。

渡したシートインデックスが存在しない場合、コマンドは空の文字列を返します。

インデックスは 0 起点です。

例題

ドキュメント内の 3つめのシートの名前を取得します:

```
$sheetName:=VP Get sheet name("ViewProArea";2)
```

参照

[VP Get sheet index](#)

VP Get sheet options

VP Get sheet options (*vpAreaName* : Text {; *sheet* : Integer }) : Object

引数	タイプ		説明
vpAreaName	Text	->	4D View Pro エリアフォームオブジェクト名
sheet	Integer	->	シートのインデックス (省略した場合はカレントシート)
戻り値	Object	<-	シートオプションのオブジェクト

説明

`VP Get sheet options` コマンドは、`vpAreaName` で指定したエリア内の、カレントのシートオプションを格納したオブジェクトを返します。

`vpAreaName` には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

任意の `sheet` 引数として、シートのインデックス (0 起点) を渡すことで、スプレッドシートを指定することができます。省略された場合、または `vk current sheet` を渡した場合、カレントスプレッドシートが使用されます。

返されるオブジェクト

コマンドは、利用可能なシートオプションのカレント値をすべて格納したオブジェクトを返します。オプションの値は、ユーザーあるいは [VP SET SHEET OPTIONS](#) コマンドによって変更される可能性があります。

オプション一覧については、[シートオプション](#) を参照ください。

例題

```
$options:=VP Get sheet options("ViewProArea")
If($options.colHeaderVisible) // カラムヘッダーが表示状態の場合
    ...
End if
```

参照

[4D VIEW PRO SHEET OPTIONS](#)

[VP SET SHEET OPTIONS](#)

VP Get show print lines

`VP Get show print lines (vpAreaName : Text {; sheet : Integer }) : Boolean`

引数	タイプ		説明
vpAreaName	Text	->	4D View Pro フォームオブジェクト名
sheet	Integer	<-	シートのインデックス
ファンクションの戻り値	Boolean	<-	印刷線が表示状態であれば <code>true</code> 、それ以外は <code>false</code>

説明

`VP Get show print lines` コマンドは、印刷プレビューの線が表示状態であれば `true` を、非表示であれば `false` を返します。

`vpAreaName` には、4D View Pro エリアの名前を渡します。

`sheet` には、ターゲットシートのインデックスを渡します。`sheet` が省略された場合、コマンドはカレントシートに対して適用されます。

インデックスは 0 起点です。

例題

以下のコードは、ドキュメントの印刷プレビュー線の表示状態をチェックします:

```

var $result : Boolean
$result:=VP Get show print lines("ViewProArea";1)

```

参照

VP SET SHOW PRINT LINES

VP Get spans

VP Get spans (*rangeObj* : Object) : Object

引数	タイプ		説明
<i>rangeObj</i>	Object	->	レンジオブジェクト
戻り値	Object	<-	指定レンジ内にあるセル結合のオブジェクト

説明

VP Get spans コマンドは、*rangeObj* で指定したレンジ内にあるセル結合を取得します。

rangeObj 引数で、セル結合を取得したいレンジを指定します。*rangeObj* にセル結合が含まれない場合には、空のレンジが返されます。

例題

ドキュメント内の結合セルにおいて、テキストを中央揃えに変更します：

		First quarter			Second quarter		
		Jan	Feb	Mar	Apr	May	Jun
South area	John						
	Sahra						
	Dixie						

```

// すべてのセル結合を検索します
$range:=VP Get spans(VP All("ViewProArea"))

// テキストを中央揃えにします
getStyle:=New object("vAlign";vk vertical align center;"hAlign";vk horizontal align center)
VP SET CELL STYLE($range;$style)

```

参照

VP ADD SPAN

VP REMOVE SPAN

VP Get stylesheet

VP Get stylesheet (*vpAreaName* : Text ; *styleName* : Text { ; *scope* : Integer }) : Object

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro フォームオブジェクト名
<i>styleName</i>	Text	->	スタイルの名前
<i>scope</i>	Integer	->	ターゲットのスコープ (デフォルト = カレントシート)
戻り値	Object	<-	スタイルシートオブジェクト

説明

VP Get stylesheet コマンドは、*styleName* で指定した、定義済のプロパティ値を格納したスタイルシートオブジェクトを返します。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

styleName には、取得するスタイルシートの名前を渡します。

任意の *scope* 引数を使用することで、スタイルシートをどこから取得するかを指定することができます。シートインデックス (0 起点) か、以下の定数のいずれかを渡すことができます：

- `vk current sheet`
- `vk workbook`

例題

以下のコードは：

```
$style:=VP Get stylesheet("ViewProArea";"GreenDashDotStyle")
```

カレントシートの *GreenDashDotStyle* スタイルオブジェクトを返します：

```
{
  backColor:green,
  borderBottom:{color:green,style:10},
  borderLeft:{color:green,style:10},
  borderRight:{color:green,style:10},
  borderTop:{color:green,style:10}
}
```

参照

[4D View Pro Style Objects and Style Sheets](#)

[VP ADD STYLESHEET](#)

[VP Get stylesheets](#)

[VP REMOVE STYLESHEET](#)

VP Get stylesheets

VP Get stylesheets (*vpAreaName* : Text { ; *scope* : Integer }) : Collection

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro フォームオブジェクト名
<i>scope</i>	Integer	->	ターゲットのスコープ (デフォルト = カレントシート)
戻り値	Collection	<-	スタイルシートオブジェクトのコレクション

説明

VP Get stylesheets コマンドは、*scope* で指定されたスコープにおいて定義されているスタイルシートのコレクションを返します。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

任意の *scope* 引数を使用することで、スタイルシートをどこに定義するかを指定することができます。シートインデックス (0 起点) か、以下の定数のいずれかを渡すことができます：

- `vk current sheet`
- `vk workbook`

例題

以下のコードは、カレントシート内にある全スタイルオブジェクトのコレクションを返します:

```
$styles:=VP Get stylesheets("ViewProArea")
```

カレントシートが 2つのスタイルオブジェクトを使用していた場合:

```
[  
 {  
   backColor:green,  
   borderLeft:{color:green,style:10},  
   borderTop:{color:green,style:10},  
   borderRight:{color:green,style:10},  
   borderBottom:{color:green,style:10},  
   name:GreenDashDotStyle  
 },  
 {  
   backColor:red,  
   textIndent:10,  
   name:RedIndent  
 }  
 ]
```

参照

[VP ADD STYLESHEET](#)

[VP Get stylesheet](#)

[VP REMOVE STYLESHEET](#)

VP Get value

VP Get value (*rangeObj* : Object) : Object

引数	タイプ		説明
<i>rangeObj</i>	Object	->	レンジオブジェクト
戻り値	Object	<-	セルの値を格納したオブジェクト

説明

VP Get value コマンドは、指定されたセルレンジからセルの値を取得します。

rangeObj 引数で、値を取得したいレンジを指定します。

返されるオブジェクト

返されるオブジェクトには `value` プロパティと、JS日付値の場合に返される `time` プロパティが格納されます:

プロパティ	タイプ	説明
<code>value</code>	Integer, Real, Boolean, Text, Date	<i>rangeObj</i> レンジの値 (ただし時間型を除く)
<code>time</code>	Real	値が js 日付型の場合、時間値 (秒単位)

返されるオブジェクトに日付または時間が含まれている場合、これは "日付時間"として扱われ、以下のように補完されます:

- 時間値 - 日付部分は DD/MM/YYYY フォーマットの、1899年12月30日 (30/12/1899) として補完されます。
- 日付値 - 時間部分は HH:MM:SS フォーマットの、真夜中 (00:00:00) として補完されます。

rangeObj のレンジが複数セルあるいは複数レンジを含んでいる場合、最初のセルの値が返されます。セルが空の場合には、コマンドは null オブジェクトを返します。

例題

```
$cell:=VP Cell("ViewProArea";5;2)
$value:=VP Get value($cell)
If(Value type($value.value)=Is text)
    VP SET TEXT VALUE($cell;New object("value";Uppercase($value.value)))
End if
```

参照

[VP Get values](#)
[VP SET VALUE](#)
[VP SET VALUES](#)

VP Get values

VP Get values (*rangeObj* : Object) : Collection

引数	タイプ		説明
<i>rangeObj</i>	Object	->	レンジオブジェクト
戻り値	Collection	<-	値のコレクション

説明

VP Get values コマンドは、*rangeObj* で指定したレンジの値をすべて取得します。

rangeObj 引数で、値を取得したいレンジを指定します。*rangeObj* のレンジが複数レンジを指定している場合、最初のレンジのみが使用されます。

VP Get values によって返されるコレクションは、2次元構造のコレクションです:

- 第1レベルのコレクションの各要素は行を表し、値のサブコレクションを格納しています。
- 各サブコレクションはその行のセル値を格納しています。値は整数、実数、布尔、テキスト、Null のいずれかです。値が日付または時間の場合は、以下のプロパティを持つオブジェクトとして返されます:

プロパティ	タイプ	説明
value	Date	セルの値 (時間部分を除く)
time	Real	値が js 日付型の場合、時間値 (秒単位)

日付または時間は 日付時間 (datetime) として扱われ、以下のように補完されます:

- 時間値 - 日付部分は 1899年12月30日として補完されます。
- 日付値 - 時間部分は真夜中 (00:00:00:000) として補完されます。

例題

C4 から G6 までの値を取得します:

	A	B	C	D	E	F	G
1							
2			1	2	3	FALSE	
3							
4			4	5	hello	world	
5			6	7	8	9	
6			29/05/2019 0:00:42				
7							

```

$result:=VP Get values(VP Cells("ViewProArea";2;3;5;3))
// $result[0]=[4,5,null,hello,world]
// $result[1]=[6,7,8,9,null]
// $result[2]=[null,{time:42,value:2019-05-29T00:00:00.000Z},null,null,null]

```

参照

[VP Get formulas](#)

[VP Get value](#)

[VP SET FORMULAS](#)

[VP SET VALUES](#)

VP Get workbook options

VP Get workbook options (*vpAreaName* : Text) : Object

引数	タイプ	説明
<i>vpAreaName</i>	Text	-> 4D View Pro フォームオブジェクト名
戻り値	Object	<- ワークブックオプションを格納したオブジェクト

説明

VP Get workbook options コマンドは、*vpAreaName* で指定したエリアのワークブックオプションをすべて格納したオブジェクトを返します。

vpAreaName には、4D View Pro エリアの名前を渡します。

返されるオブジェクトには、ワークブック内のワークブックオプションの値（デフォルト値および変更値）がすべて格納されています。

ワークブックオプションの一覧については [VP SET WORKBOOK OPTIONS の説明](#) を参照ください。

例題

```

var $workbookOptions : Object
$workbookOptions:=VP Get workbook options("ViewProArea")

```

参照

[VP SET WORKBOOK OPTIONS](#)

I

VP IMPORT DOCUMENT

VP IMPORT DOCUMENT (*vpAreaName* : Text ; *filePath* : Text { ; *paramObj* : Object})

引数	タイプ	説明
<i>vpAreaName</i>	Text	-> 4D View Pro フォームオブジェクト名
<i>filePath</i>	Text	-> ドキュメントのパス名
<i>paramObj</i>	Object	-> 読み込みのオプション

説明

VP IMPORT DOCUMENT コマンドは、*vpAreaName* で指定した 4D View Pro エリアに、*filePath* 引数のドキュメントを読み込んで表示させま

す。エリア内に挿入されていたデータがあれば、それらは読み込んだドキュメントによって置換されます。

`vpAreaName` には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

`filePath` には読み込むドキュメントのパスとファイル名を渡します。以下のフォーマットがサポートされています：

- 4D View Pro ドキュメント (拡張子 ".4vp")
- Microsoft Excel 形式 (拡張子 ".xlsx")
- テキスト形式のドキュメント (拡張子 ".txt", ".csv", ドキュメントは UTF-8 形式である必要あり)

ドキュメントの拡張子が認識される拡張子 (.4vp や .xlsx など) ではなかった場合、ドキュメントはテキスト形式であると見なされます。ドキュメントが Project フォルダーと同階層に置かれている場合を除き、フルパスを渡す必要があります (同階層に置かれている場合にはファイル名のみを渡すことができます)。

Microsoft Excel 形式のファイルを 4D View Pro ドキュメントに読み込む場合、一部の設定が失われる可能性があります。GrapeCity にある[一覧](#)にて、設定を検証することができます。

`filePath` 引数が無効だった場合や、対象ファイルが存在しなかったり、壊れたりしている場合には、エラーが返されます。

任意の `paramObj` 引数を渡すことで、読み込まれるドキュメントのプロパティを定義することができます:

引数		タイプ	説明
formula		object	書き出し終了時に実行させるコールバックメソッド名。メソッドは Formula コマンドと組み合わせて使用する必要があります。 コールバックメソッドの渡し方 を参照ください。
password		text	Microsoft Excel のみ (任意) - MS Excel ドキュメントの保護に使用されているパスワード。
csvOptions		object	CSV読み込みのオプション
	range	object	書き出されるデータの、最初のセルを格納しているセルレンジ。指定されたレンジがセルレンジではない場合、レンジの最初のセルが使用されます。
	rowDelimiter	text	行の区切り文字。渡されなかった場合、区切り文字は 4D によって自動的に定義されます。
	columnDelimiter	text	カラムの区切り文字。デフォルト: ","

CSV形式および、ユーザー定義区切りの値 (DSV) については、こちらの [Wikipedia の記事 \(英文\)](#) を参照ください。

例題 1

フォームが開かれたときに、ディスク上に保存されているデフォルトの 4D View Pro ドキュメントを読み込みます:

```
C_TEXT($docPath)
If(Form event code=On VP Ready) // 4D View Pro エリアの読み込みが完了しています
    $docPath:="C:\\Bases\\ViewProDocs\\MyExport.4VP"
    VP IMPORT DOCUMENT("VPArea";$docPath)
End if
```

例題 2

パスワードで保護されている Microsoft Excel ドキュメントを 4D View Pro エリアに読み込みます:

```
$o:=New object
$o.password:="excel123"

VP IMPORT DOCUMENT("ViewProArea";"c:\\tmp\\excelfilefile.xlsx";$o)
```

例題 3

カンマ (",") を区切り文字として使用している .txt ファイルを読み込みます:

```
"Clark","Kent"  
"Bruce","Wayne"  
"Barry","Allen"  
"Peter","Parker"  
"Tony","Stark"
```

```
$params:=New object  
$params.range:=VP Cells("ViewProArea";0;0;2;5)  
VP IMPORT DOCUMENT("ViewProArea";"c:\\import\\my-file.txt";New object("csvOptions";$params))
```

このようになります:

The screenshot shows a spreadsheet application interface. The ribbon menu includes FILE, HOME, INSERT, FORMULAS, DATA, VIEW, and SETTINGS. The HOME tab is selected, displaying various toolbar icons for file operations, clipboard, fonts, alignment, numbers, cell type, styles, cells, and editing. The formula bar shows 'A1' and 'Clark'. The main workspace displays a table with 5 rows and 2 columns. Row 1 contains 'Clark' in cell A1 and 'Kent' in cell B1. Rows 2 through 5 contain 'Bruce' and 'Wayne', 'Barry' and 'Allen', 'Peter' and 'Parker', and 'Tony' and 'Stark' respectively. The bottom status bar indicates 'Ready' and '100% zoom'.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Clark	Kent											
2	Bruce	Wayne											
3	Barry	Allen											
4	Peter	Parker											
5	Tony	Stark											

参照

[VP EXPORT DOCUMENT](#)
[VP NEW DOCUMENT](#)

VP IMPORT FROM OBJECT

VP IMPORT FROM OBJECT (vpAreaName : Text { ; viewPro : Object})

引数	タイプ		説明
vpAreaName	Text	->	4D View Pro フォームオブジェクト名
viewPro	Object	->	4D View Pro オブジェクト

説明

VP IMPORT FROM OBJECT コマンドは、vpAreaName で指定した 4D View Pro エリアに viewPro の 4D View Pro オブジェクトを読み込んで表示させます。エリア内に挿入されていたデータがあれば、それらは読み込んだオブジェクトのデータで置換されます。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

`viewPro` には有効な 4D View Pro オブジェクトを渡します。このオブジェクトは手動で作成するほか、[VP Export to object](#) を使って取得することができます。4D View Pro オブジェクトについての詳細は [4D View Pro オブジェクト](#) を参照ください。

`viewPro` オブジェクトが無効な場合には、エラーが返されます。

例題

オブジェクトフィールドに保存してあるスプレッドシートを読み込みます:

```
QUERY([VPWorkBooks];[VPWorkBooks]ID=10)
VP IMPORT FROM OBJECT("ViewProArea1";[VPWorkBooks]SPBook)
```

参照

[VP Export to object](#)

VP INSERT COLUMNS

`VP INSERT COLUMNS` (*rangeObj* : Object)

引数	タイプ		説明
<i>rangeObj</i>	Object	->	レンジオブジェクト

説明

`VP INSERT COLUMNS` コマンドは、*rangeObj* 引数で指定したレンジにカラムを挿入します。

rangeObj には、開始カラム（新しいカラムが挿入される場所を指定するカラム）と挿入するカラムの数を格納したオブジェクトを渡します。挿入するカラムの数が省略された場合（定義されていない場合）、カラムは 1列だけ挿入されます。

新しいカラムは、*rangeObj* 引数で指定した開始カラムの直前（すぐ左側）に挿入されます。

例題

2番目のカラムの前にカラムを3列挿入します:

```
VP INSERT COLUMNS(VP Column("ViewProArea";1;3))
```

このようになります:

Before insertion

	A	B	C	D	E	F	G	H	I	J
1	The	quick	brown	fox	jumped	over	the	lazy	dog	
2										

After insertion

	A	B	C	D	E	F	G	H	I	J	K	L
1	The				quick	brown	fox	jumped	over	the	lazy	dog
2												

参照

[VP DELETE COLUMNS](#)

[VP DELETE ROWS](#)
[VP INSERT ROWS](#)

VP INSERT ROWS

VP INSERT ROWS (*rangeObj* : Object)

引数	タイプ		説明
<i>rangeObj</i>	Object	->	レンジオブジェクト

説明

VP INSERT ROWS コマンドは、*rangeObj* で指定したレンジに行を挿入します。

rangeObj には、開始行（新しい行が挿入される場所を指定する行）と挿入する行数を格納したオブジェクトを渡します。挿入する行数が省略された場合（定義されていない場合）には、1行だけ挿入されます。

新しい行は、*rangeObj* 引数で指定した開始行の直前（すぐ上）に挿入されます。

例題

先頭行の前に 3 行挿入します：

```
VP INSERT ROWS(VP Row("ViewProArea";0;3))
```

このようになります：

Before insertion

	A	B	C	D	E	F	G	H	I	J
1	The	quick	brown	fox	jumped	over	the	lazy	dog	
2										
3										
4										
<										

After insertion

	A	B	C	D	E	F	G	H	I	J
1										
2										
3										
4	The	quick	brown	fox	jumped	over	the	lazy	dog	
<										

参照

[VP DELETE COLUMNS](#)
[VP DELETE ROWS](#)
[VP INSERT COLUMNS](#)

M

VP MOVE CELLS

▶ [履歴](#)

VP MOVE CELLS (*originRange* : Object ; *targetRange* : Object ; *options* : Object)

引数	タイプ		説明
originRange	Object	->	移動 (コピー) 元のセルレンジ
targetRange	Object	->	値・書式・フォーミュラの移動 (コピー) 先レンジ
options	Object	->	追加のオプション

説明

`VP MOVE CELLS` コマンドは、*originRange* の値・書式・フォーミュラを *targetRange* に移動またはコピーします。

originRange と *targetRange* は異なる 4D View Pro エリアを参照することができます。

originRange には、移動またはコピーする値・書式・フォーミュラが格納されているレンジオブジェクトを渡します。 *originRange* が結合レンジの場合は、最初のものだけが使用されます。

targetRange には、値・書式・フォーミュラのコピー先または移動先であるターゲットレンジを渡します。

options は、複数のプロパティを持ちます:

プロパティ	タイプ	説明														
copy	Boolean	<i>originRange</i> のセルの値・書式・フォーミュラをコマンド実行後に削除するかどうかを指定します: <ul style="list-style-type: none"> 削除するには <code>false</code> (デフォルト)。 保持するには <code>true</code>。 														
pasteOptions	Longint	ペーストする内容を指定します。 どうる値: <table border="1"> <thead> <tr> <th>値</th> <th>説明</th> </tr> </thead> <tbody> <tr> <td><code>vk clipboard options all</code> (デフォルト)</td> <td>値・フォーマット・フォーミュラを含むすべてのデータオブジェクトをペーストします。</td> </tr> <tr> <td><code>vk clipboard options formatting</code></td> <td>フォーマットだけをペーストします。</td> </tr> <tr> <td><code>vk clipboard options formulas</code></td> <td>フォーミュラだけをペーストします。</td> </tr> <tr> <td><code>vk clipboard options formulas and formatting</code></td> <td>フォーミュラとフォーマットをペーストします。</td> </tr> <tr> <td><code>vk clipboard options values</code></td> <td>値だけをペーストします。</td> </tr> <tr> <td><code>vk clipboard options value and formatting</code></td> <td>値とフォーマットをペーストします。</td> </tr> </tbody> </table>	値	説明	<code>vk clipboard options all</code> (デフォルト)	値・フォーマット・フォーミュラを含むすべてのデータオブジェクトをペーストします。	<code>vk clipboard options formatting</code>	フォーマットだけをペーストします。	<code>vk clipboard options formulas</code>	フォーミュラだけをペーストします。	<code>vk clipboard options formulas and formatting</code>	フォーミュラとフォーマットをペーストします。	<code>vk clipboard options values</code>	値だけをペーストします。	<code>vk clipboard options value and formatting</code>	値とフォーマットをペーストします。
値	説明															
<code>vk clipboard options all</code> (デフォルト)	値・フォーマット・フォーミュラを含むすべてのデータオブジェクトをペーストします。															
<code>vk clipboard options formatting</code>	フォーマットだけをペーストします。															
<code>vk clipboard options formulas</code>	フォーミュラだけをペーストします。															
<code>vk clipboard options formulas and formatting</code>	フォーミュラとフォーマットをペーストします。															
<code>vk clipboard options values</code>	値だけをペーストします。															
<code>vk clipboard options value and formatting</code>	値とフォーマットをペーストします。															

[ワークブックオプション](#) で定義されている貼り付けオプションが考慮されます。

例題

内容・値・書式・フォーミュラをコピーします:

```

var $originRange; $targetRange; $options : Object
$originRange:=VP Cells("ViewProArea"; 0; 0; 2; 5)
$targetRange:=VP Cells("ViewProArea"; 4; 0; 2; 5)
$options:=New object
$options.copy:=True
$options.pasteOptions:=vk clipboard options all
VP MOVE CELLS($originRange; $targetRange; $options)

```

参照

[VP Copy to object](#)
[VP PASTE FROM OBJECT](#)
[VP SET WORKBOOK OPTIONS](#)

N

VP Name

VP Name (*vpAreaName* : Text ; *rangeName* : Text { ; *scope* : Integer }) : Object

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro フォームオブジェクト名
<i>rangeName</i>	Text	->	既存のレンジ名
<i>scope</i>	Integer	->	レンジの場所 (省略時はカレントシート)
戻り値	Object	<-	<i>rangeName</i> のレンジオブジェクト

説明

VP Name コマンドは、命名レンジを参照する新しいレンジオブジェクトを返します。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

rangeName には、既存のセルレンジ名を渡します。

任意の *scope* 引数として、*rangeName* のレンジが属するスプレッドシートを指定することができます。省略された場合はデフォルトでカレントスプレッドシートが使用されます。以下の定数を使用することでカレントのスプレッドシートあるいはワークブック全体を明示的に選択することができます：

- `vk current sheet`
- `vk workbook`

例題

"Total" という名前のレンジに値を渡します：

```
// B5 のセルを "Total" と命名します
VP ADD RANGE NAME(VP Cell("ViewProArea";1;4);"Total")
$name:=VP Name("ViewProArea";" Total")
VP SET NUM VALUE($name;285;"$#,###.00")
```

参照

[VP ADD RANGE NAME](#)
[VP ALL](#)
[VP Cell](#)
[VP Cells](#)
[VP Column](#)
[VP Combine ranges](#)
[VP Get names](#)
[VP REMOVE NAME](#)
[VP Row](#)

VP NEW DOCUMENT

VP NEW DOCUMENT (*vpAreaName* : Text)

引数	タイプ		説明
vpAreaName	Text	->	4D View Pro フォームオブジェクト名

説明

`VP NEW DOCUMENT` コマンドは、`vpAreaName` で指定した 4D View Pro エリアに、新規のデフォルトドキュメントを読み込んで表示させます。エリア内に挿入されていたデータがあれば、それらは新規の空ドキュメントによって置換されます。

`vpAreaName` には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

例題

"myVPArea" フォームオブジェクトに空ドキュメントを表示します：

```
VP NEW DOCUMENT("myVPArea")
```

参照

[VP IMPORT DOCUMENT](#)

O

VP Object to font

`VP Object to font (fontObj : Object) : Text`

引数	タイプ		説明
fontObj	Object	->	フォントオブジェクト
戻り値	Text	<-	フォントのショートハンド文字列

説明

`VP Object to font` コマンドは、`fontObj` 引数で指定したフォントオブジェクトからフォントのショートハンド文字列を返します。

`fontObj` には、フォントプロパティを格納するオブジェクトを渡します。以下のオブジェクトプロパティがサポートされています：

プロパティ	タイプ	説明	とりうる値	必須
family	text	フォントを指定します。	標準の、あるいは一般的なフォントファミリー。例: "Arial", "Helvetica", "serif", "arial,sans-serif"	○
size	text	フォントのサイズを定義します。 "font-size/line-height" の形で line-height を font-size に追加することもできます: 例: "15pt/20pt"	以下のいずれかの単位を伴う数値: <ul style="list-style-type: none"> ● "em", "ex", "%", "px", "cm", "mm", "in", "pt", "pc", "ch", "rem", "vh", "vw", "vmin", "vmax" あるいは、以下の定数のいずれか 1つ: <ul style="list-style-type: none"> ● <code>vk font size large</code> ● <code>vk font size larger</code> ● <code>vk font size x large</code> ● <code>vk font size xx large</code> ● <code>vk font size small</code> ● <code>vk font size smaller</code> ● <code>vk font size x small</code> ● <code>vk font size xx small</code> 	○
style	text	フォントのスタイル。	<ul style="list-style-type: none"> ● <code>vk font style italic</code> ● <code>vk font style oblique</code> 	×
variant	text	スモールキャピタルのフォントを定義します。	<ul style="list-style-type: none"> ● <code>vk font variant small caps</code> 	×
weight	text	フォントの太さを定義します。	<ul style="list-style-type: none"> ● <code>vk font weight 100</code> ● <code>vk font weight 200</code> ● <code>vk font weight 300</code> ● <code>vk font weight 400</code> ● <code>vk font weight 500</code> ● <code>vk font weight 600</code> ● <code>vk font weight 700</code> ● <code>vk font weight 800</code> ● <code>vk font weight 900</code> ● <code>vk font weight bold</code> ● <code>vk font weight bolder</code> ● <code>vk font weight lighter</code> 	×

このオブジェクトは [VP Font to object](#) コマンドで作成することができます。

返されるショートハンド文字列は、たとえば [VP SET CELL STYLE](#) を使って、セルの "font" プロパティに割り当てることができます。

例題

```
$cellStyle:=VP Get cell style($range)

$font:=VP Font to object($cellStyle.font)
$font.style:=vk font style oblique
$font.variant:=vk font variant small caps
$font.weight:=vk font weight bolder

$cellStyle.font:=VP Object to font($font)
// $cellStyle.font には "bolder oblique small-caps 16pt arial" が格納されます
```

参照

4D View Pro Style Objects and Style Sheets

[VP Font to object](#)

[VP SET CELL STYLE](#)

[VP SET DEFAULT STYLE](#)

P

VP PASTE FROM OBJECT

▶ [履歴](#)

VP PASTE FROM OBJECT (*rangeObj* : Object ; *dataObject* : Object {; *options* : Longint})

引数	タイプ	説明
<i>rangeObj</i>	Object	-> セルレンジオブジェクト
<i>dataObject</i>	Object	-> ペーストするデータを格納したオブジェクト
<i>options</i>	Longint	-> ペーストする内容を指定します

説明

VP PASTE FROM OBJECT コマンドは、*dataObject* のコンテンツ・スタイル・フォーミュラを *rangeObj* セルレンジオブジェクトにペーストします。

rangeObj には、値・フォーマット・フォーミュラをペーストする先のセルレンジオブジェクトを渡します。 *rangeObj* が複数のセルを参照している場合は、最初のセルだけが使用されます。

dataObject には、ペーストしたい値・フォーマット・フォーミュラを格納しているオブジェクトを渡します。

任意の *options* 引数を渡して、セルレンジにペーストする内容を指定することができます。とりうる値:

定数	説明
<code>vk clipboard options all</code>	値・フォーマット・フォーミュラを含むすべてのデータオブジェクトをペーストします。
<code>vk clipboard options formatting</code>	フォーマットだけをペーストします。
<code>vk clipboard options formulas</code>	フォーミュラだけをペーストします。
<code>vk clipboard options formulas and formatting</code>	フォーミュラとフォーマットをペーストします。
<code>vk clipboard options values</code>	値だけをペーストします。
<code>vk clipboard options value and formatting</code>	値とフォーマットをペーストします。

[ワークブックオプション](#) で定義されている貼り付けオプションが考慮されます。

dataObject に存在しない要素を *options* で指定した場合 (例: フォーミュラ) コマンドはなにもしません。

例題

[VP Copy to object](#) の例題を参照ください。

参照

[VP Copy to object](#)

[VP MOVE CELLS](#)

[VP Get workbook options](#)

[VP SET WORKBOOK OPTIONS](#)

VP PRINT

VP PRINT (*vpAreaName* : Text { ; *sheet* : Integer })

引数	タイプ		説明
vpAreaName	Text	->	4D View Pro フォームオブジェクト名
sheet	Integer	->	シートのインデックス (省略した場合はカレントシート)

説明

`VP PRINT` コマンドは、`vpAreaName` 引数で指定したエリアを印刷する印刷ダイアログウィンドウを開きます。

`vpAreaName` には、印刷する 4D View Pro エリアの名前を渡します。コマンドによって、システムの印刷ダイアログウィンドウが開かれ、プリンターを指定したりページプロパティを定義したりすることができます。

印刷ダイアログウィンドウで定義されるプロパティはプリンター用紙のためのもので、4D View Pro エリアの印刷プロパティではありません。4D View Pro エリアの印刷プロパティは [VP SET PRINT INFO](#) コマンドで定義されます。プリンターと 4D View Pro エリアの両プロパティが一致することが強く推奨されます。そうでない場合、ドキュメントが期待通りに印刷されない可能性があります。

任意の `sheet` 引数として、シートのインデックス (0 起点) を渡すことで、印刷するスプレッドシートを指定することができます。省略された場合はデフォルトでカレントシートが使用されます。以下の定数を使用することでカレントのスプレッドシートあるいはワークブック全体を明示的に選択することができます:

- `vk current sheet`
- `vk workbook`

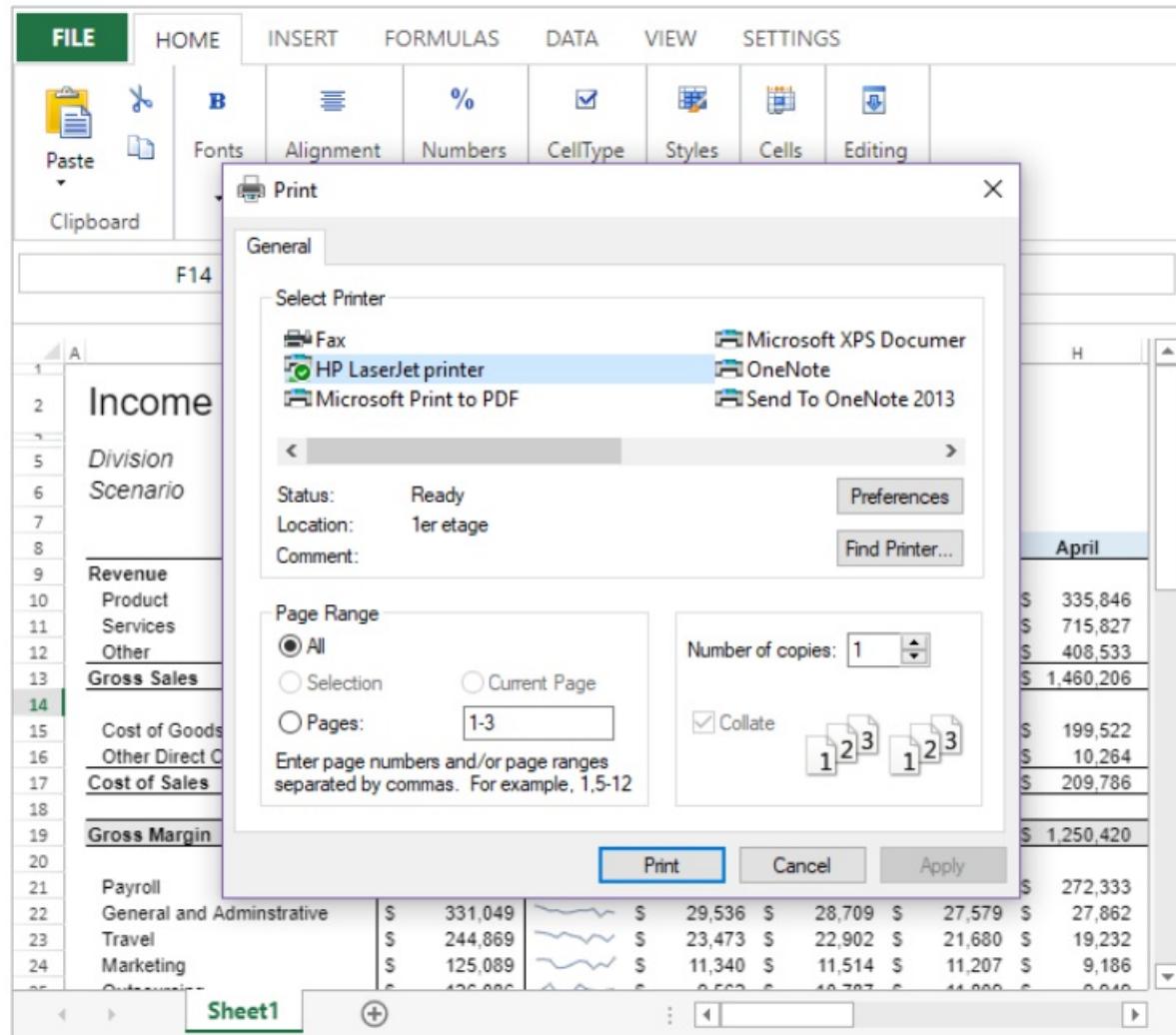
- 4D View Pro エリアは、`VP PRINT` コマンドによってのみ印刷可能です。
- 4Dコマンドの印刷テーマのコマンドは、`VP PRINT` ではサポートされません。
- このコマンドは、最終的なエンドユーザーが個別に印刷を実行することを想定しています。自動化印刷ジョブについては、[VP EXPORT DOCUMENT](#) コマンドで 4D View Pro エリアを PDF に書き出すことが推奨されます。

例題

以下のコードは:

```
VP PRINT("myVPArea")
```

印刷ダイアログウィンドウを開きます:



参照

[VP EXPORT DOCUMENT](#)
[VP SET PRINT INFO](#)

R

VP RECOMPUTE FORMULAS

VP RECOMPUTE FORMULAS (vpAreaName : Text)

引数	タイプ	説明
vpAreaName	Text	-> 4D View Pro フォームオブジェクト名

説明

VP RECOMPUTE FORMULAS コマンドは、*vpAreaName* 引数で指定したエリアの全フォーミュラを即座に評価します。デフォルトでは、4D はフォーミュラを挿入時、読み込み時、そして書き出し時のタイミングで自動計算します。VP RECOMPUTE FORMULAS コマンドを使用すると、任意のタイミングで強制的に計算を実行することができます(例: フォーミュラに変更が加えられた場合、またはフォーミュラがデータベースへの呼び出しを格納している場合など)。コマンドは [VP FLUSH COMMANDS](#) コマンドを実行することで、実行保留されていたコマンドをすべて実行し、コマンドバッファをクリアします。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

VP RECOMPUTE FORMULAS コマンドを使用する前に [VP SUSPEND COMPUTING](#) コマンドが実行されていないようにしてください。

例題

ワークブック内の全フォーミュラを更新します:

```
VP RECOMPUTE FORMULAS("ViewProArea")
```

参照

[VP RESUME COMPUTING](#)
[VP SUSPEND COMPUTING](#)

VP REMOVE NAME

VP REMOVE NAME (*vpAreaName* : Text ; *name* : Text { ; *scope* : Integer })

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro フォームオブジェクト名
<i>name</i>	Text	->	削除する命名レンジまたは命名フォーミュラの名前
<i>scope</i>	Integer	->	ターゲットのスコープ (デフォルト=カレントシート)

説明

`VP REMOVE NAME` コマンドは、*name* の命名レンジまたは命名フォーミュラを、定義された *scope* のスコープから削除します。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

name には、削除したい命名レンジあるいは命名フォーミュラの名前を渡します。

scope 引数を使用することで、命名レンジまたは命名フォーミュラをどこから削除するのか、指定することができます。その際、シートのインデックス (0 起点) を渡すか、以下の定数のいずれかを渡します:

- `vk current sheet`
- `vk workbook`

例題

```
$range:=VP Cell("ViewProArea";0;0)
VP ADD RANGE NAME("Total1";$range)

VP REMOVE NAME("ViewProArea";"Total1")
$formula:=VP Get formula by name("ViewProArea";"Total1")
// $formula=null
```

参照

[VP Name](#)

VP REMOVE SHEET

VP REMOVE SHEET (*vpAreaName* : Text ; *index*: Integer)

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro フォームオブジェクト名
<i>index</i>	Integer	->	削除するシートのインデックス

参照

[VP ADD SHEET](#)

説明

`VP REMOVE SHEET` コマンドは、`vpAreaName` 引数で指定したエリアにロードされているドキュメントから、`index` 引数で指定したインデックスのシートを削除します。

`vpAreaName` には、4D View Pro エリアの名前を渡します。

`index` 引数には、削除したいシートのインデックスを渡します。`index` 引数に渡したインデックスが存在しない場合、このコマンドは何もしません。

インデックスは 0 起点です。

例題

ドキュメントには現在 3つのシートがあります:



3つ目のシートを削除します:

```
VP REMOVE SHEET("ViewProArea";2)
```



VP REMOVE SPAN

`VP REMOVE SPAN (rangeObj : Object)`

引数	タイプ		説明
<code>rangeObj</code>	<code>Object</code>	<code>-></code>	レンジオブジェクト

説明

`VP REMOVE SPAN` コマンドは、`rangeObj` で指定したレンジ内のセル結合を解除します。

`rangeObj` には、セル結合しているレンジのオブジェクトを渡します。レンジ内の結合セルは個別セルに分割されます。

例題

ドキュメントのセル結合をすべて解除します:

		First quarter			Second quarter		
		Jan	Feb	Mar	Apr	May	Jun
South area	John						
	Sahra						
	Dixie						

```
// すべてのセル結合を探します  
$span:=VP Get spans(VP All("ViewProArea"))
```

```
// すべてのセル結合を解除します  
VP REMOVE SPAN($span)
```

結果:

		First quart			Second qu		
		Jan	Feb	Mar	Apr	May	Jun
South area	John						
	Sahra						
	Dixie						

参照

[VP ADD SPAN](#)

[VP Get spans](#)

VP REMOVE STYLESHEET

VP REMOVE STYLESHEET (*vpAreaName* : Text ; *styleName* : Text { ; *scope* : Integer })

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro フォームオブジェクト名
<i>styleName</i>	Text	->	削除するスタイルの名前
<i>scope</i>	Integer	->	ターゲットのスコープ (デフォルト = カレントシート)

説明

`VP REMOVE STYLESHEET` コマンドは、*vpAreaName* 引数で指定したエリアから、*styleName* で指定したスタイルシートを削除します。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

styleName 引数には、削除するスタイルシートの名前を渡します。

任意の *scope* 引数を使用することで、スタイルシートをどこから削除するかを指定することができます。シートインデックス (0 起点) か、以下の定数のいずれかを渡すことができます:

- `vk current sheet`
- `vk workbook`

例題

カレントシートから、*GreenDashDotStyle* スタイルオブジェクトを削除します:

```
VP REMOVE STYLESHEET("ViewProArea";"GreenDashDotStyle")
```

参照

[VP ADD STYLESHEET](#)

[VP Get stylesheet](#)

[VP Get stylesheets](#)

VP RESET SELECTION

VP RESET SELECTION (*vpAreaName* : Text { ; *sheet* : Integer })

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro フォームオブジェクト名
<i>sheet</i>	Integer	->	シートのインデックス (省略した場合はカレントシート)

説明

VP RESET SELECTION コマンドは、すべてのセル選択を解除し、その結果カレントセレクション（またはアクティブセル）がなくなります。

4D View Pro コマンドに対して定義されているデフォルトのアクティブセル（A1 セル）は残ります。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

任意の *sheet* 引数として、シートのインデックス（0 起点）を渡すことで、定義されるレンジが属するスプレッドシートを指定することができます。省略された場合はデフォルトでカレントスプレッドシートが使用されます。以下の定数を使用することでカレントのスプレッドシートを明示的に選択することができます：

- `vk current sheet`

例題

セル選択（アクティブセルと選択セル）をすべて解除します：

```
VP RESET SELECTION("myVPArea")
```

参照

[VP ADD SELECTION](#)

[VP Get active cell](#)

[VP Get selection](#)

[VP SET ACTIVE CELL](#)

[VP SET SELECTION](#)

[VP SHOW CELL](#)

VP RESUME COMPUTING

VP RESUME COMPUTING (*vpAreaName* : Text)

引数	タイプ	説明
<i>vpAreaName</i>	Text	-> 4D View Pro フォームオブジェクト名

説明

VP RESUME COMPUTING コマンドは、*vpAreaName* 引数で指定したエリア内の計算を再開します。

このコマンドは 4D View Pro 内の計算機能を再開します。計算停止中におこなった編集に影響されたフォーミュラは更新され、**VP RESUME COMPUTING** 実行後に追加されるフォーミュラは計算されます。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

4D View Pro 計算機能は停止/再開アクションを内部的にカウントしています。そのため、**VP RESUME COMPUTING** コマンドの実行数は、**VP SUSPEND COMPUTING** コマンドの実行数と釣り合っていかなければなりません。

例題

[VP SUSPEND COMPUTING](#) の例題を参照ください。

参照

[VP RECOMPUTE FORMULAS](#)

[VP SUSPEND COMPUTING](#)

VP Row

VP Row (*vpAreaName* : Text; *row* : Integer { ; *rowCount* : Integer { ; *sheet* : Integer } }) : Object

引数	タイプ		説明
vpAreaName	Text	->	4D View Pro フォームオブジェクト名
row	Integer	->	行のインデックス
rowCount	Integer	->	行数
sheet	Integer	->	シートのインデックス (省略した場合はカレントシート)
戻り値	Object	<-	行のレンジオブジェクト

説明

VP Row コマンドは、特定行、あるいは複数行を参照する新しいレンジオブジェクトを返します。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

任意の *columnCount* には、レンジに含まれるカラム数を指定することができます。この引数として行のインデックス (0 起点) を渡します。*columnCount* 引数は 0 より大きい値でなければなりません。

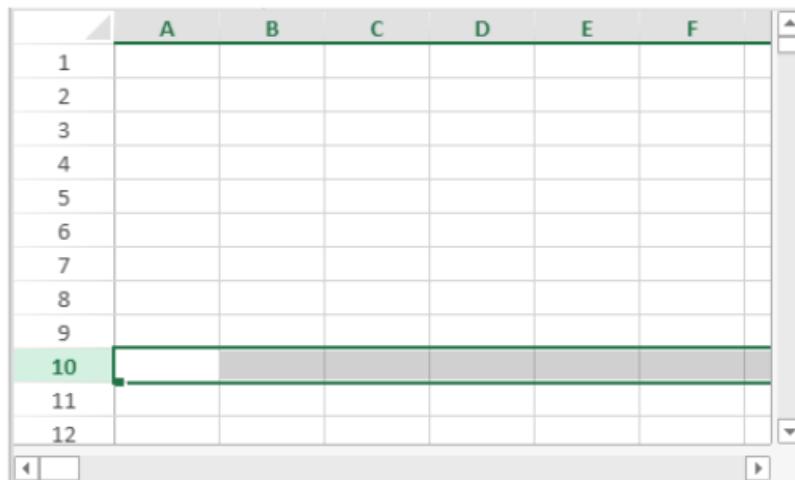
レンジが複数行にわたる場合には、任意の *rowCount* 引数も併せて使用します。*rowCount* 引数は 0 より大きい値でなければなりません。省略時、デフォルトで値は 1 に設定されます。

任意の *sheet* 引数として、シートのインデックス (0 起点) を渡すことで、定義されるレンジが属するスプレッドシートを指定することができます。省略された場合はデフォルトでカレントスプレッドシートが使用されます。以下の定数を使用することでカレントのスプレッドシートを明示的に選択することができます:

- `vk current sheet`

例題

以下に表示されている (カレントスプレッドシートの) 行に対するレンジオブジェクトを定義します:



以下のように書くことができます:

```
$row:=VP Row("ViewProArea";9) // 10行目
```

参照

[VP All](#)
[VP Cell](#)
[VP Cells](#)
[VP Column](#)
[VP Combine ranges](#)
[VP Name](#)

VP ROW AUTOFIT

VP ROW AUTOFIT (*rangeObj* : Object)

引数	タイプ		説明
rangeObj	Object	->	レンジオブジェクト

説明

VP ROW AUTOFIT コマンドは、*rangeObj* 引数のレンジ内の行を、そのコンテンツに応じて自動的にリサイズします。

`rangeObj` 引数として、サイズを自動調整したい行を格納しているレンジオブジェクトを渡します。

例題

以下の行では、テキストを正しく表示できていません:

```
VP ROW AUTOFIT(VP Row("ViewProArea";1;2))
```

結果：

参照

VP Column autofit

VP Run offscreen area

VP Run offscreen area (*parameters* : Object) : Mixed

引数	タイプ		説明
parameters	Object	->	オフスクリーンエリアの属性を格納するオブジェクト
戻り値	Mixed	<-	.onEvent オブジェクトの .result プロパティ、または値を返さない場合には Null

説明

VP Run offscreen area コマンド、メモリ内にオフスクリーンエリアを作成し、これを利用して 4D View Pro エリアのコマンドやファンクションを処理することができます。

`parameters` オブジェクトには、以下の任意のプロパティのいずれかを渡します。これらのプロパティは `onEvent` コールバックメソッド内において `This` コマンドを介して利用可能であり、そのインスタンスを参照することができます：

プロパティ	タイプ	説明
area	text	オフスクリーンエリアの名前。省略時あるいは null の場合、一般的な名前（例："OffscreenArea1"）が割り当てられます。
onEvent	object (フォーミュラ)	<p>オフスクリーンエリアの準備ができたときに実行されるコールバックメソッド。以下のいずれかを渡すことができます：</p> <ul style="list-style-type: none"> • クラスの <code>onEvent</code> 関数 • <code>Formula</code> オブジェクト <p>デフォルトでは、コールバックメソッドは、<code>On VP Ready</code>, <code>On Load</code>, <code>On Unload</code>, <code>On End URL Loading</code>, <code>On URL Loading Error</code>, <code>On VP Range Changed</code>, または <code>On Timer</code> イベントで呼び出されます。</p> <p>コールバックメソッドを使用して 4D View Pro フォームオブジェクト変数 にアクセスすることができます。</p>
autoQuit	boolean	<p>True（デフォルト値）の場合、<code>On End URL Loading</code> または <code>On URL Loading Error</code> イベントが起きた際にはコマンドがフォーミュラの実行を中止します。</p> <p>False の場合、<code>onEvent</code> コールバックメソッド内で <code>CANCEL</code> あるいは <code>ACCEPT</code> コマンドを使用する必要があります。</p>
timeout	number	イベントが何も生成されない場合にエリアが自動的に閉まるまでの最大時間（秒単位）。0 に設定した場合、エリアは自動的には閉まりません。デフォルト値：60
result	混合	処理の結果（あれば）
<customProperty>	混合	<code>onEvent</code> コールバックメソッドで利用可能なカスタムの属性。

以下のプロパティは、必要に応じてコマンドによって自動的に追加されます：

プロパティ	タイプ	説明
timeoutReached	boolean	タイムアウトを超えた場合に true の値で追加されます

オフスクリーンエリアは、`VP Run offscreen area` コマンドの実行中にしか利用できません。実行が終わるとエリアは自動的に消去されます。

コールバックメソッドでは、以下のコマンドを使用することができます：

- `ACCEPT`
- `CANCEL`
- `SET TIMER`
- `WA Evaluate JavaScript`
- `WA EXECUTE JAVASCRIPT FUNCTION`

例題 1

オフスクリーンの 4D View Pro エリアを作成し、そこからセルの値を取得します：

```

// cs.OffscreenArea クラス宣言
Class constructor ($path : Text)
    This.filePath:=$path

// この関数はオフスクリーンエリアの各イベントごとに呼び出されます
Function onEvent()
    Case of
        :(FORM Event.code=On VP Ready)
            VP IMPORT DOCUMENT(This.area;This.filePath)
            This.result:=VP Get value(VP Cell(This.area;6;22))

            ALERT("The G23 cell contains the value: "+String(This.result))
    End case

```

OffscreenArea コールバックメソッドの内容は以下の通りです:

```

$o:=cs.OffscreenArea.new()
$result:=VP Run offscreen area($o)

```

例題 2

大きなドキュメントをオフスクリーンで読み込み、計算の評価が完了するの待ってドキュメントを PDF として書き出します:

```

//cs.OffscreenArea クラス宣言
Class constructor($pdfPath : Text)
    This.pdfPath:=$pdfPath
    This.autoQuit:=False
    This.isWaiting:=False

Function onEvent()
    Case of
        :(FORM Event.code=On VP Ready)
        // ドキュメントの読み込み
            VP IMPORT DOCUMENT(This.area;$largeDocument4VP)
            This.isWaiting:=True

        // 計算が完了したかを検証するタイマーをスタートさせます
        // この間に "On VP Range Changed" イベントが発生した場合、タイマーはリスタートされます
        // 時間はコンピューターの設定に応じて定義されなければなりません
            SET TIMER(60)

        :(FORM Event.code=On VP Range Changed)
        // 計算の完了を感じし、 タイマーを再スタートさせます
            If(This.isWaiting)
                SET TIMER(60)
            End if

        :(FORM Event.code=On Timer)
        // この時点以降、他の 4D View コマンドを呼び出してもタイマーが再スタートしないようにします
            This.isWaiting:=False

        // タイマーを停止します
            SET TIMER(0)

        // PDF 書き出しを開始します
            VP EXPORT DOCUMENT(This.area;This.pdfPath;New object("formula";Formula(ACCEPT)))

        :(FORM Event.code=On URL Loading Error)
            CANCEL
    End case
    // この間に "On VP Range Changed" イベントが発生した場合、タイマーはリスタートされます
    // 時間はコンピューターの設定に応じて定義されなければなりません
        SET TIMER(60)

        :(FORM Event.code=On VP Range Changed)
        // 計算の完了を感じし、 タイマーを再スタートさせます
            If(This.isWaiting)
                SET TIMER(60)
            End if

        :(FORM Event.code=On Timer)
        // この時点以降、他の 4D View コマンドを呼び出してもタイマーが再スタートしないようにします
            This.isWaiting:=False

        // タイマーを停止します
            SET TIMER(0)

        // PDF 書き出しを開始します
            VP EXPORT DOCUMENT(This.area;This.pdfPath;New object("formula";Formula(ACCEPT)))

        :(FORM Event.code=On URL Loading Error)
            CANCEL
    End case

```

OffscreenArea コールバックメソッドの内容は以下の通りです:

```
$o:=cs.OffscreenArea.new()
$result:=VP Run offscreen area($o)
```

参照

[ブログ記事 \(英文\): End of document loading](#)

S

VP SET ACTIVE CELL

VP SET ACTIVE CELL (*rangeObj* : Object)

引数	タイプ		説明
<i>rangeObj</i>	Object	->	レンジオブジェクト

説明

VP SET ACTIVE CELL コマンドは、指定したセルをアクティブセルにします。

rangeObj 引数には、単独のセルを格納するレンジオブジェクトを渡します ([VP Cell 参照](#))。*rangeObj* 引数のレンジが単独セルのレンジでない場合、あるいは複数レンジを指定している場合、最初のレンジの先頭セルが使用されます。

例題

カラム D、行 5 のセルをアクティブセルに設定します:

```
$activeCell:=VP Cell("myVPArea";3;4)
VP SET ACTIVE CELL($activeCell)
```

	A	B	C	D	E	F	G
1							
2	Month	Area	Mr. Smith	Ms. Johnson	Ms. Williams	Mr. Jones	M
3	January	South	1898	1857	1651	1448	
4		East	4859	4857	2548	4876	
5		North	2458	1524	6150	4987	
6		West	5787	1580	3975	4878	
7	Total		15002	9818	14324	16189	
8	February	South	6668	4374	17495	9999	
9		East	5955	1677	7944	9400	
10		North	1000	6722	2195	2777	
11		West	6896	8355	7195	2058	
12	Total		20519	21128	34829	24234	
13	March	South	2577	2000	6185	2704	
14		East	4859	4857	2548	4876	
15		North	2458	1524	6150	4987	
16		West	5787	1580	3975	4878	
17	Total		15681	9961	18858	17445	

参照

[VP ADD SELECTION](#)
[VP Get active cell](#)
[VP Get selection](#)
[VP RESET SELECTION](#)

[VP SET SELECTION](#)
[VP SHOW CELL](#)

VP SET ALLOWED METHODS

VP SET ALLOWED METHODS (*methodObj* : Object)

引数	タイプ		説明
methodObj	Object	->	4D View Pro エリアでの実行を許可するメソッド

互換性

より高い柔軟性のため、4D View Pro エリアから呼び出せる 4D フォーミュラを指定できる [VP SET CUSTOM FUNCTIONS](#) コマンドの使用が推奨されます。 [VP SET CUSTOM FUNCTIONS](#) が呼び出された場合、[VP SET ALLOWED METHODS](#) の呼び出しは無視されます。 [VP SET CUSTOM FUNCTIONS](#) と [VP SET ALLOWED METHODS](#) のどちらも呼び出されていない場合、4D View Pro は 4D の汎用コマンド [SET ALLOWED METHODS](#) もサポートしますが、汎用コマンドの使用は推奨されません。

説明

`VP SET ALLOWED METHODS` コマンドは、4D View Pro フォーミュラから呼び出し可能なプロジェクトメソッドを指定します。このコマンドは、呼び出し後のセッション中に初期化される 4D View Pro エリアすべてに対して適用されます。同じセッション中において異なる設定で初期化するために、複数回呼び出すこともできます。

セキュリティ上の理由により、`VP SET ALLOWED METHODS` コマンドを実行していない場合のデフォルトでは、4D View Pro エリアにおいてはメソッドの呼び出しは許可されません（ただし、4D の汎用的な `SET ALLOWED METHODS` コマンドが呼び出されていた場合を除きます）。許可されていないのメソッドをフォーミュラ内で使用した場合には、4D View Pro エリアに #NAME? 許可されていないのメソッドをフォーミュラ内で使用した場合には、4D View Pro エリアに #NAME? エラーが表示されます。

methodObj には、4D View Pro エリア内で定義したいファンクションの名前をプロパティとして格納しているオブジェクトを渡します：

プロパティ			タイプ	説明
<functionName>			Object	カスタムファンクションの名前。<customFunction> は、4D View Pro フォーミュラで表示するカスタムファンクションの名前を定義します（スペースは使用できません）
	method		Text	(必須) 許可する既存の 4D プロジェクトメソッドの名前
	parameters		Object の Collection	引数のコレクション（メソッド内で定義されている順）
	[].name		Text	<functionName> 用に表示する引数の名前。 注: 引数の名前にスペースを含めることはできません。
	[].type		Number	引数の型。サポートされている型: <ul style="list-style-type: none"> ● Is Boolean ● Is date ● Is Integer ● Is object ● Is real ● Is text ● Is time <p>省略時のデフォルトでは、値は型と一緒に渡されますが、日付と時間の値に関してはオブジェクトとして送られます（引数の章を参照ください）。type が Is object の場合、そのオブジェクトは VP Get value によって返されるオブジェクトと同じ構造を持ちます。</p>
	summary		Text	4D View Pro に表示するファンクションの説明
	minParams		Number	引数の最小の数
	maxParams		Number	引数の最大の数。ここに parameters の length より大きな値を渡すことによって、デフォルトの型を持つ "任意の" 引数を宣言することができるようになります。

例題

4D View Pro エリアにおいて、2つのメソッドを許可します：

```
C_OBJECT($allowed)
$allowed:=New object // コマンドに渡す引数

$allowed.Hello:=New object // "Hello" という名前の 1つ目の簡単なファンクションを作成します
$allowed.Hello.method:="My_Hello_Method" // 4Dメソッドを設定します
$allowed.Hello.summary:="Hello prints hello world"

$allowed.Byebye:=New object // "Byebye" という名前の、引数を受け付ける 2つ目のファンクションを作成
$allowed.Byebye.method:="My_ByeBye_Method"
$allowed.Byebye.parameters:=New collection
$allowed.Byebye.parameters.push(New object("name","Message","type";Is text))
$allowed.Byebye.parameters.push(New object("name","Date","type";Is date))
$allowed.Byebye.parameters.push(New object("name","Time","type";Is time))
$allowed.Byebye.summary:="Byebye prints a custom timestamp"
$allowed.Byebye.minParams:=3
$allowed.Byebye.maxParams:=3

VP SET ALLOWED METHODS($allowed)
```

このコードが実行されたあと、定義されたファンクションは 4D View Pro フォーミュラで使用することができるようになります：

	A	B	C	D	E
1	=BYEBYE(
2		BYEBYE(Message; Date; Time)			
3		Summary			
4		Byebye prints a custom timestamp			
5					

インデックスは 0 起点です。

参照

[4D functions](#)

[VP SET CUSTOM FUNCTIONS](#)

VP SET BOOLEAN VALUE

VP SET BOOLEAN VALUE (*rangeObj* : Object ; *boolValue* : Boolean)

引数	タイプ		説明
<i>rangeObj</i>	Object	->	レンジオブジェクト
<i>boolValue</i>	Boolean	->	設定するブール値

説明

VP SET BOOLEAN VALUE コマンドは、指定のセルレンジにブール値を割り当てます。

rangeObj には、値を割り当てるセルのレンジ (たとえば [VP Cell](#) あるいは [VP Column](#) で作成されたレンジ) を渡します。 *rangeObj* 引数に複数のセルが含まれる場合、指定された値はそれぞれのセルに対して繰り返し割り当てられます。

boolValue 引数には、*rangeObj* のセルレンジに割り当てるブール値 (true あるいは false) を渡します。

例題

```
// セルの値を false に設定
VP SET BOOLEAN VALUE(VP Cell("ViewProArea";3;2);False)
```

参照

[VP SET VALUE](#)

VP SET BORDER

VP SET BORDER (*rangeObj* : Object ; *borderStyleObj* : Object ; *borderPosObj* : Object)

引数	タイプ	説明
<i>rangeObj</i>	Object	-> レンジオブジェクト
<i>borderStyleObj</i>	Object	-> 境界線スタイルを格納したオブジェクト
<i>borderPosObj</i>	Object	-> 境界線の位置を格納したオブジェクト

説明

VP SET BORDER コマンドは、*rangeObj* のレンジに *borderStyleObj* および *borderPosObj* で定義される境界線スタイルを適用します。

rangeObj 引数には、境界線スタイルを適用したいセルのレンジを渡します。*rangeObj* 引数に複数のセルが含まれる場合、[VP SET BORDER](#) で適用される境界線は、*rangeObj* のレンジ全体を一つのセルとして適用されます (これに対し、[VP SET CELL STYLE](#) コマンドでは*rangeObj* 引数のレンジに含まれる個々のセルに対し境界線が適用されます)。スタイルシートがすでに適用されている場合、[VP SET BORDER](#) コマンドは

rangeObj のレンジに対してすでに適用されていた境界線設定を上書きします。

borderStyleObj 引数を使用すると、境界線のスタイルを定義することができます。*borderStyleObj* 引数は、以下のプロパティをサポートしています：

プロパティ	タイプ	説明	とりうる値
color	text	境界線のカラーを定義します。デフォルト = black	CSSカラー "#rrggbb" シンタックス (推奨シンタックス)、CSSカラー "rgb(r,g,b)" シンタックス (代替シンタックス)、CSSカラーネーム (代替シンタックス)
style	Integer	境界線のスタイルを定義します。デフォルト = empty。	<ul style="list-style-type: none">● <code>vk line style dash dot</code>● <code>vk line style dash dot dot</code>● <code>vk line style dashed</code>● <code>vk line style dotted</code>● <code>vk line style double</code>● <code>vk line style empty</code>● <code>vk line style hair</code>● <code>vk line style medium</code>● <code>vk line style medium dash dot</code>● <code>vk line style medium dash dot dot</code>● <code>vk line style medium dashed</code>● <code>vk line style slanted dash dot</code>● <code>vk line style thick</code>● <code>vk line style thin</code>

borderStyleObj の境界線スタイルの位置 (どこに境界線を引くか) は *borderPosObj* 引数で定義します：

プロパティ	タイプ	説明
all	boolean	境界線スタイルはすべての境界に適用されます。
left	boolean	境界線スタイルは左の境界に適用されます。
top	boolean	境界線スタイルは上の境界に適用されます。
right	boolean	境界線スタイルは右の境界に適用されます。
bottom	boolean	境界線スタイルは下の境界に適用されます。
outline	boolean	境界線スタイルは外側の境界にのみ適用されます。
inside	boolean	境界線スタイルは内側の境界にのみ適用されます。
innerHorizontal	boolean	境界線スタイルは内側の横の境界にのみ適用されます。
innerVertical	boolean	境界線スタイルは内側の縦の境界にのみ適用されます。

例題 1

以下のコードは、レンジ全体の外周に境界線を生成します：

```
$border:=New object("color";"red";"style";vk line style thick)
$option:=New object("outline";True)
VP SET BORDER(VP Cells("ViewProArea";1;1;3;3);$border;$option)
```

	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					

例題 2

以下のコードは、`VP SET BORDER` と `VP SET CELL STYLE` で境界線を設定した場合の違いを示します：

```
// VP SET BORDER で境界線を設定します
$border:=New object("color";"red";"style";vk line style thick)
$option:=New object("outline";True)
VP SET BORDER(VP Cells("ViewProArea";1;1;3;3);$border;$option)

// VP SET CELL STYLE を使用して境界線を設定します
$cellStyle:=New object
$cellStyle.borderBottom:=New object("color";"blue";"style";vk line style thick)
$cellStyle.borderRight:=New object("color";"blue";"style";vk line style thick)
VP SET CELL STYLE(VP Cells("ViewProArea";4;4;3;3);$cellStyle)
```

	A	B	C	D	E	F	G	H
1								
2								
3								
4								
5								
6								
7								
8								

参照

[VP SET CELL STYLE](#)

VP SET CELL STYLE

`VP SET CELL STYLE` (*rangeObj* : Object ; *styleObj* : Object)

引数	タイプ		説明
<i>rangeObj</i>	Object	->	レンジオブジェクト
<i>styleObj</i>	Object	->	スタイルオブジェクト

説明

`VP SET CELL STYLE` コマンドは、*styleObj* に定義されているスタイルを、*rangeObj* で定義されたセルに適用します。

rangeObj 引数には、スタイルを適用したいセルのレンジを渡します。*rangeObj* に複数のセルが含まれる場合、スタイルはそれぞれのセルに割り当てられます。

rangeObj がセルレンジではない場合、レンジの最初のセルが使用されます。

styleObj にはスタイル設定を格納したオブジェクトを渡します。既存のスタイルシートを使用することもできますし、新しいスタイルを作成することも可能です。*styleObj* に既存のスタイルシートと、追加のスタイル設定の両方が格納されている場合、既存のスタイルシートが先に適用され、その後に追加の設定が適用されます。

スタイルを削除してデフォルトのスタイル設定（あれば）に戻すには、NULL値を渡します：

- *styleObj* 引数として NULL 値を渡した場合、*rangeObj* のレンジのスタイルシートはすべて削除されます。
- 属性に NULL 値を指定すると、当該属性は *rangeObj* から削除されます。

スタイルオブジェクトとスタイルシートの詳細については、[スタイルオブジェクト](#) を参照ください。

例題

```
$style:=New object
$style.font:="8pt Arial"
$style.backColor:="Azure"
$style.foreColor:="red"
$style.hAlign:=1
$style.isVerticalText:=True
$style.borderBottom:=New object("color";"#800080";"style";vk line style thick)
$style.backgroundImage:=Null // 特定の属性を削除します

VP SET CELL STYLE(VP Cell("ViewProArea";1;1);$style)
```

	A	B	C
1			
2		H e l l o	
3		W o r l d	

参照

[VP ADD STYLESHEET](#)
[VP Font to object](#)
[VP Get cell style](#)
[VP Object to font](#)
[VP SET BORDER](#)
[VP SET DEFAULT STYLE](#)

VP SET COLUMN ATTRIBUTES

VP SET COLUMN ATTRIBUTES (*rangeObj* : Object ; *propertyObj* : Object)

引数	タイプ		説明
<i>rangeObj</i>	Object	->	レンジオブジェクト
<i>propertyObj</i>	Object	->	カラムプロパティを格納したオブジェクト

説明

VP SET COLUMN ATTRIBUTES コマンドは、*rangeObj* 引数で指定したカラムに対して *propertyObj* 引数で定義されている属性を適用します。

rangeObj 引数には、レンジオブジェクトを渡します。レンジにカラムと行の両方が格納されている場合、属性は行に対してのみ適用されます。

propertyObj 引数は、*rangeObj* 引数のレンジ内のカラムに対して適用する属性を指定します。指定できる属性は以下の通りです:

プロパティ	タイプ	説明
width	number	カラムの幅 (ピクセル単位)
pageBreak	boolean	レンジ内の先頭カラムの前に改ページを挿入する場合には true、それ以外は false
visible	boolean	カラムが表示状態であれば true、それ以外は false
resizable	boolean	カラムがリサイズ可能であれば true、それ以外は false
headers	text	カラムヘッダーのテキスト

例題

2列目のカラムの幅を変更して、ヘッダーを設定します:

```
C_OBJECT($column;$properties)

$column:=VP Column("ViewProArea";1) // カラムB を取得
$properties:=New object("width";100;"header";"Hello World")

VP SET COLUMN ATTRIBUTES($column;$properties)
```

A	B	C	D	E	F	G	H	I
1	The	quick	brown	fox	jumped	over	the	lazy dog

参照

[VP Column](#)
[VP Get column attributes](#)
[VP Get row attributes](#)
[VP SET ROW ATTRIBUTES](#)

VP SET COLUMN COUNT

VP SET COLUMN COUNT (*vpAreaName* : Text , *columnCount* : Integer { , *sheet* : Integer })

引数	タイプ	説明
<i>vpAreaName</i>	Text	-> 4D View Pro フォームオブジェクト名
<i>columnCount</i>	Integer	-> カラム数
<i>sheet</i>	Integer	-> シートのインデックス (省略した場合はカレントシート)

説明

`VP SET COLUMN COUNT` コマンドは、*vpAreaName* 引数内にあるカラムの総数を定義します。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

columnCount には、カラムの総数を渡します。*columnCount* 引数は 0 より大きい値でなければなりません。

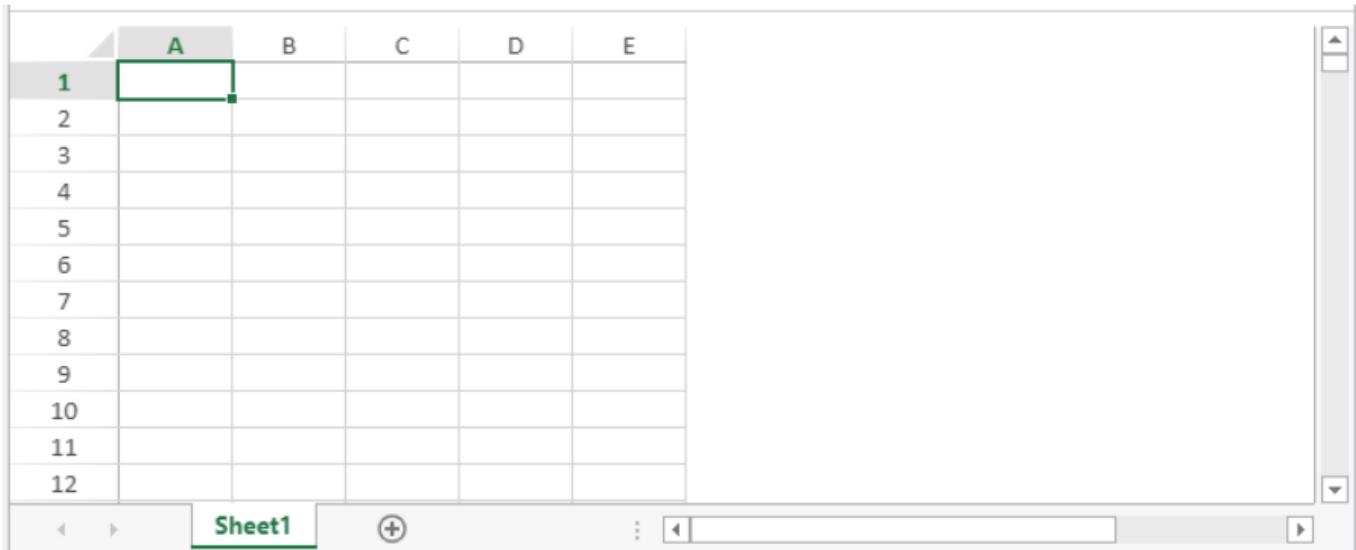
任意の *sheet* 引数として、シートのインデックス (0 起点) を渡すことで、*columnCount* が適用されるスプレッドシートを指定することができます。省略された場合はデフォルトでカレントスプレッドシートが使用されます。以下の定数を使用することでカレントのスプレッドシートを明示的に選択することができます:

- `vk current sheet`

例題

以下のコードは 4D View Pro エリア内に 5つのカラムを定義します:

```
VP SET COLUMN COUNT("ViewProArea";5)
```



参照

[VP Get column count](#)
[VP Get row count](#)
[VP SET ROW COUNT](#)

VP SET CURRENT SHEET

VP SET CURRENT SHEET (*vpAreaName* : Text ; *index* : Integer)

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro フォームオブジェクト名
<i>index</i>	Integer	<-	新しいカレントシートのインデックス

説明

`VP SET CURRENT SHEET` コマンドは、*vpAreaName* 引数で指定した View Pro エリアのカレントシートを設定します。カレントシートとは、ドキュメント内で選択されているシートのことです。

vpAreaName には、4D View Pro エリアの名前を渡します。

index 引数には、カレントシートに設定したいシートのインデックスを渡します。*index* 引数が 0 未満の場合、またはシートの総数より多い場合、コマンドは何もしません。

インデックスは 0 起点です。

例題

ドキュメントの最初のシートがカレントシートになっています:



カレントシートを第3シートに設定します:

```
VP SET CURRENT SHEET("ViewProArea";2)
```



参照

[VP Get current sheet](#)

VP SET CUSTOM FUNCTIONS

VP SET CUSTOM FUNCTIONS (*vpAreaName* : Text ; *formulaObj* : Object)

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro フォームオブジェクト名
<i>formulaObj</i>	Object	->	フォーミュラオブジェクト

説明

VP SET CUSTOM FUNCTIONS コマンドは、4D View Pro フォーミュラから直接呼び出し可能な 4D フォーミュラを指定します。カスタムのファンクションはドキュメント内に保存されていないので、VP SET CUSTOM FUNCTIONS は On Load フォームイベント内で呼び出される必要があります。

VP SET CUSTOM FUNCTIONS で指定されたフォーミュラは、最初の文字が入力されるとポップアップメニューに表示されます。詳細については [式と関数](#) を参照ください。

インデックスは 0 起点です。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

formulaObj 引数として、4D View Pro フォーミュラから呼び出し可能な 4D フォーミュラと、その追加のプロパティを格納したオブジェクトを渡します。*formulaObj* 引数の各 `customFunction` プロパティが 4D View Pro エリア内のファンクション名になります。

プロパティ			タイプ	説明
<customFunction>			Object	カスタムファンクションの名前。<customFunction> は、4D View Pro フォーミュラで表示するカスタムファンクションの名前を定義します（スペースは使用できません）
	formula		Object	4Dフォーミュラオブジェクト（必須）。Formula コマンド参照。
	parameters		Object の Collection	引数のコレクション（フォーミュラ内で定義されている順）
	[].name	Text		4D View Pro に表示する引数の名前。
	[].type	Number		<p>引数の型。サポートされている型：</p> <ul style="list-style-type: none"> ● Is Boolean ● Is date ● Is Integer ● Is object ● Is real ● Is text ● Is time <p><i>type</i>省略時、またはデフォルト値 (-1) が渡された場合、値は型と一緒に渡されますが、日付と時間の値に関してはオブジェクトとして送られます（引数の章を参照ください）。</p> <p><i>type</i> が Is object の場合、そのオブジェクトは VP Get value によって返されるオブジェクトと同じ構造を持ちます。</p>
	summary	Text		4D View Pro に表示するフォーミュラの説明
	minParams	Number		引数の最小の数
	maxParams	Number		引数の最大の数。ここに <i>parameters</i> の length より大きな値を渡すことによって、デフォルトの型を持つ "任意の" 引数を宣言できるようになります。

警告

- VP SET CUSTOM FUNCTIONS が呼び出された場合、VP SET ALLOWED METHODS コマンドにより許可されたメソッド（あれば）は同 4D View Pro エリアにおいて無視されます。
- VP SET CUSTOM FUNCTIONS が呼び出されると、4D View Pro エリアは SET TABLE TITLES や SET FIELD TITLES コマンドに基づく機能を無視します。

例題

4D View Pro エリア内で、フォーミュラオブジェクトを使用して、数値を追加/顧客の苗字/顧客の性別を取得します：

```

Case of
:(FORM Event.code=On Load)

var $o : Object
$o:=New object

// "addnum" メソッドを使用した "addnum" ファンクションを定義します
$o.addnum:=New object
$o.addnum.formula:=Formula(addnum)
$o.addnum.parameters:=New collection
$o.addnum.parameters.push(New object("name";"num1";"type";Is Integer))
$o.addnum.parameters.push(New object("name";"num2";"type";Is Integer))

// データベースフィールドから "ClientLastName" ファンクションを定義します
$o.ClientLastName:=New object
$o.ClientLastName.formula:=Formula([Customers]lastname)
$o.ClientLastName.summary:="Lastname of the current client"

// 引数を 1つ受け取る 4D式から "label" ファンクションを定義します
$o.label:=New object
$o.label.formula:=Formula(ds.Customers.$1.label)
$o.label.parameters:=New collection
$o.label.parameters.push(New object("name";"ID";"type";Is Integer))

// Define "Title" function from a variable named "Title"
$o.Title:=New object
$o.Title.formula:=Formula(Title)

VP SET CUSTOM FUNCTIONS("ViewProArea";$o)

```

End case

参照

VP SET ALLOWED METHODS

VP SET DATE TIME VALUE

VP SET DATE TIME VALUE (*rangeObj* : Object ; *dateValue* : Date ; *timeValue* : Time {; *formatPattern* : Text })

引数	タイプ		説明
<i>rangeObj</i>	Object	->	レンジオブジェクト
<i>dateValue</i>	Date	->	設定する日付値
<i>timeValue</i>	Time	->	設定する時間値
<i>formatPattern</i>	Text	->	値のフォーマット

説明

VP SET DATE TIME VALUE コマンドは、指定されたセルレンジに日付・時間値を割り当てます。

rangeObj には、値を割り当てたいセルのレンジ (たとえば [VP Cell](#) あるいは [VP Column](#) で作成されたレンジ) を渡します。 *rangeObj* 引数に複数のセルが含まれる場合、指定された値はそれぞれのセルに対して繰り返し割り当てられます。

dateValue 引数に、*rangeObj* 引数のレンジに割り当てたい日付値を指定します。

timeValue 引数に、*rangeObj* 引数のレンジに割り当てたい時間値 (秒単位) を指定します。

任意の *formatPattern* 引数は、*dateValue* および *timeValue* 引数に対するパターンを定義します。 パターンおよびフォーマット文字に関しての情報については、[日付と時間のフォーマット](#) の章を参照してください。

例題

```
// セルの値をカレントの日付と時間に設定  
VP SET DATE TIME VALUE(VP Cell("ViewProArea";6;2);Current time;Current date;vk pattern full date time)  
  
// セルの値を 12月18日に設定  
VP SET DATE TIME VALUE(VP Cell("ViewProArea";3;9);!2024-12-18!;?14:30:10?;vk pattern sortable date time)
```

参照

[4D View Pro cell format](#)

[VP SET DATE VALUE](#)

[VP SET TIME VALUE](#)

[VP SET VALUE](#)

VP SET DATE VALUE

VP SET DATE VALUE (*rangeObj* : Object ; *dateValue* : Date { ; *formatPattern* : Text })

引数	タイプ		説明
<i>rangeObj</i>	Object	->	レンジオブジェクト
<i>dateValue</i>	Date	->	設定する日付値
<i>formatPattern</i>	Text	->	値のフォーマット

説明

VP SET DATE VALUE コマンドは、指定されたセルレンジに日付値を割り当てます。

rangeObj には、値を割り当てるセルのレンジを渡します。 *rangeObj* 引数に複数のセルが含まれる場合、指定された値はそれぞれのセルに対して繰り返し割り当てられます。

dateValue 引数に、*rangeObj* 引数のレンジに割り当てる日付値を指定します。

任意の *formatPattern* 引数は、*dateValue* 引数に対するパターンを定義します。カスタムのフォーマット、または以下の定数のいずれかを渡します：

定数	説明	デフォルト US パターン
vk pattern long date	ISO 8601 フォーマットの完全な日付。	"dddd, dd MMMM yyyy"
vk pattern month day	ISO 8601 フォーマットの月と日付。	"MMMM dd"
vk pattern short date	省略形の ISO 8601 フォーマットの日付。	"MM/dd/yyyy"
vk pattern year month	ISO 8601 フォーマットの年と月。	"yyyy MMMM"

パターンおよびフォーマット文字についての情報については、[日付と時間のフォーマット](#) の章を参照してください。

例題

```
// セルの日付をカレントの日付に設定します  
VP SET DATE VALUE(VP Cell("ViewProArea";4;2);Current date))  
  
// セルの値を、指定されたフォーマットの特定の日付に設定します  
VP SET DATE VALUE(VP Cell("ViewProArea";4;4);Date("12/25/94");"d/m/yy ")  
VP SET DATE VALUE(VP Cell("ViewProArea";4;6);!2005-01-15!;vk pattern month day)
```

参照

4D View Pro cell format

VP SET DATE TIME VALUE

VP SET VALUE

VP SET DEFAULT STYLE

VP SET DEFAULT STYLE (*vpAreaName* : Text ; *styleObj* : Object { ; *sheet* : Integer })

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro フォームオブジェクト名
<i>styleObj</i>	Object	->	スタイルオブジェクト
<i>sheet</i>	Integer	->	シートインデックス (デフォルト=カレントシート)

説明

VP SET DEFAULT STYLE コマンドは、*sheet* で指定したシートに対して、*styleObj* 引数のスタイルをデフォルトスタイルとして定義します。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

styleObj にはスタイル設定を格納したオブジェクトを渡します。既存のスタイルシートを使用することもできますし、新しいスタイルを作成することも可能です。詳細については、[スタイルオブジェクト](#) を参照ください。

任意の *sheet* 引数として、シートのインデックスを渡すことで、スタイルが定義されるスプレッドシートを指定することができます。省略された場合はデフォルトでカレントスプレッドシートが使用されます。以下の定数を使用することでカレントのスプレッドシートを明示的に選択することができます：

- `vk current sheet`

例題

```
$style:=New object
$style.hAlign:=vk horizontal align left
$style.font:="12pt papyrus"
$style.backColor:="#E6E6FA" // 薄い紫色

VP SET DEFAULT STYLE("myDoc";$style)
```

A screenshot of a spreadsheet application showing a single cell B2 containing the text "Hello World!". The cell is highlighted with a light purple background. The column header A and row header 1 are visible at the top left. The row number 2 is also visible above the cell.

	A	B	C
1			
2		Hello World!	
3			
4			
5			
6			
7			
8			

参照

[VP ADD STYLESHEET](#)

[VP Font to object](#)

[VP Get default style](#)

[VP Object to font](#)

[VP SET BORDER](#)

[VP SET CELL STYLE](#)

VP SET FIELD

VP SET FIELD (*rangeObj* : Object ; *field* : Pointer { ; *formatPattern* : Text })

引数	タイプ		説明
<i>rangeObj</i>	Object	->	レンジオブジェクト
フィールド	Pointer	->	仮想ストラクチャーのフィールドへの参照
<i>formatPattern</i>	Text	->	フィールドのフォーマット

説明

`VP SET FIELD` コマンドは、指定されたセルレンジに、4Dデータベースの仮想フィールドを割り当てます。

rangeObj には、値を割り当てるセルのレンジを渡します。 *rangeObj* には、値を割り当てるセルのレンジ（`VP Cell` で作成されたレンジ）を渡します。

field 引数は、*rangeObj* のレンジに対して割り当てられる 4Dデータベースの [仮想フィールド](#) を指定します。フォーミュラバーには、*field* の仮想ストラクチャー名が表示されます。*rangeObj* に含まれるセルに既存のコンテンツがあった場合、そのコンテンツは *field* で上書きされます。

任意の *formatPattern* 引数は、*field* 引数に対するパターンを定義します。有効な [カスタムフォーマット](#) を渡すことができます。

例題

```
VP SET FIELD(VP Cell("ViewProArea";5;2);->[TableName]Field)
```

参照

[VP SET VALUE](#)

VP SET FORMULA

VP SET FORMULA (*rangeObj* : Object ; *formula* : Text { ; *formatPattern* : Text })

引数	タイプ		説明
<i>rangeObj</i>	Object	->	レンジオブジェクト
<i>formula</i>	Text	->	フォーミュラまたは 4Dメソッド
<i>formatPattern</i>	Text	->	フィールドのフォーマット

説明

`VP SET FORMULA` コマンドは、指定されたセルレンジにフォーミュラまたは 4Dメソッドを割り当てます。

rangeObj には、値を割り当てるセルのレンジ（たとえば `VP Cell` あるいは `VP Column` で作成されたレンジ）を渡します。 *rangeObj* には、値を割り当てるセルのレンジを渡します。

formula 引数に、*rangeObj* 引数のレンジに割り当てるフォーミュラまたは 4Dメソッド名を指定します。4Dメソッドを使用する場合、そのメソッドは [SET ALLOWED METHODS](#) コマンドで許可されている必要があります。

任意の *formatPattern* 引数は、*formula* に対する [パターン](#) を定義します。

rangeObj 内のフォーミュラは、空の文字列 ("") で置き換えることで削除することができます。

例題 1

```
VP SET FORMULA(VP Cell("ViewProArea";5;2);"SUM($A$1:$C$10)")
```

例題 2

フォーミュラを削除します:

```
VP SET FORMULA(VP Cell("ViewProArea";5;2); "")
```

参照

[Cell format](#)

[VP Get Formula](#)

[VP SET FORMULAS](#)

[VP SET VALUE](#)

VP SET FORMULAS

`VP SET FORMULAS (rangeObj : Object ; formulasCol : Collection)`

引数	タイプ		説明
rangeObj	Object	->	セルレンジオブジェクト
formulasCol	Collection	->	フォーミュラのコレクション

説明

`VP SET FORMULAS` コマンドは、指定のセルレンジから開始してフォーミュラのコレクションを割り当てていきます。

rangeObj には、フォーミュラを割り当てるセルのレンジ (`VP Cell` で作成されたレンジ) を渡します。*rangeObj* のレンジが複数レンジを指定している場合、最初のレンジのみが使用されます。

formulasCol 引数は 2次元構造のコレクションです:

- 第1レベルのコレクションは、フォーミュラのサブコレクションを格納しています。それぞれのサブコレクションは行を定義します。
- それぞれのサブコレクションは行におけるセルの値を定義します。値は、セルに割り当てるフォーミュラを格納したテキスト要素でなくてはなりません。 > 4Dメソッドを使用する場合、そのメソッドは `SET ALLOWED METHODS` コマンドで許可されている必要があります。

rangeObj 内のフォーミュラは、空の文字列 ("") で置き換えることで削除することができます。

例題 1

```
$formulas:=New collection
$formulas.push(New collection("MAX(B11,C11,D11)";"myMethod(G4)")) // 一行目
$formulas.push(New collection("SUM(B11:D11)";"AVERAGE(B11:D11)")) // 二行目

VP SET FORMULAS(VP Cell("ViewProArea";6;3);$formulas) // フォーミュラをセルに設定します
```

myMethod:

```
$0:=$1*3.33
```

	A	B	C	D	E	F	G	H	I	J
1	Production Cost Analysis									
2	Retail Price Calculations									
3		Product 1	Product 2	Product 3						
4		\$0.98	\$0.74	\$0.18		Highest Prod Cost	\$4.42	\$14.72	Retail Price	
5		\$0.23	\$0.83	\$0.46		Total Production Cost	\$11.38	\$3.79	Average Product Cost	
6		\$0.77	\$0.09	\$0.53						
7		\$0.74	\$0.11	\$0.47						
8		\$0.59	\$0.93	\$0.63						
9		\$0.85	\$0.65	\$0.34						
10		\$0.26	\$0.25	\$0.75						
11	Per Product Total	\$4.42	\$3.60	\$3.36						

例題 2

フォーミュラを削除します:

```
$formulas:=New collection
$formulas.push(New collection("", ""))
$formulas.push(New collection("", ""))
VP SET FORMULAS(VP Cell("ViewProArea";0;0);$formulas) // セルに割り当てます
```

参照

[VP Get Formulas](#)

[VP GET VALUES](#)

[VP SET VALUES](#)

VP SET FROZEN PANES

VP SET FROZEN PANES (*vpAreaName* : Text ; *paneObj* : Object { ; *sheet* : Integer })

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro フォームオブジェクト名
<i>paneObj</i>	Object	->	固定化されたカラムと行についての情報を格納したオブジェクト
<i>sheet</i>	Integer	->	シートのインデックス (省略した場合はカレントシート)

説明

VP SET FROZEN PANES コマンドは、*vpAreaName* 引数で指定した View Pro エリア内の、*paneObj* 引数のカラムと行の固定化ステータスを設定します。固定化されたカラムと行は固定された位置に表示され続け、ドキュメントの他の部分がスクロールされても移動しません。そのカラムと行が固定化されていることを示すために、太い実線が表示されます。実線の位置は、固定化されたカラムまたは行がシートのどこにあるかによって変わります:

- 左または右にあるカラム: シートの左側にあるカラムについては、実線は最後に固定化されたカラム（最も右のカラム）の右側に表示されます。シートの右側に表示されているカラムについては、実線は最初に固定化されたカラム（最も左のカラム）の左側に表示されます。
- 上または下にある行: シートの上部にある行については、実線は最後に固定化された行（最も下の行）の下側に表示されます。シートの下部に表示されている行については、実線は最初に固定化された行（最も上の行）の上側に表示されます。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

paneObj には、固定化するカラムと行を定義するオブジェクトを渡します。以下のカラムまたは行のプロパティの値にゼロを設定すると、そのプロパティをリセット（固定解除）します。プロパティが 0以下 の値に設定された場合、コマンドは何もしません。以下のものを渡すことができます：

プロパティ	タイプ	説明
columnCount	Integer	シートの左側にある固定化されたカラム
trailingColumnCount	Integer	シートの右側にある固定化されたカラム
rowCount	Integer	シートの上側にある固定化された行
trailingRowCount	Integer	シートの下側にある固定化された行

任意の *sheet* 引数として、シートのインデックス（0 起点）を渡すことで、定義されるレンジが属するスプレッドシートを指定することができます。省略された場合はデフォルトでカレントスプレッドシートが使用されます。以下の定数を使用することでカレントのスプレッドシートを明示的に選択することができます：

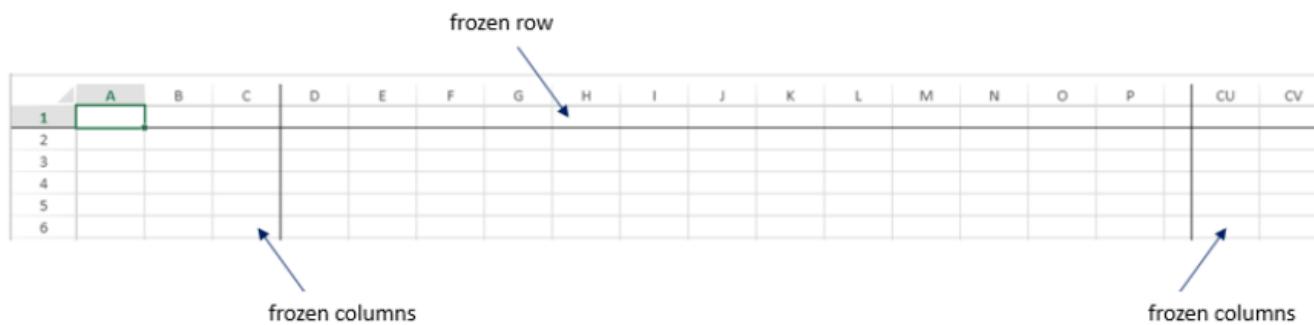
- `vk current sheet`

例題

左側の最初の 3つのカラム、右側にある 2つのカラム、そして最初の行を固定化します：

```
C_OBJECT($panes)
$panes:=New object
$panes.columnCount:=3
$panes.trailingColumnCount:=2
$panes.rowCount:=1

VP SET FROZEN PANES("ViewProArea";$panes)
```



参照

[VP Get frozen panes](#)

VP SET NUM VALUE

VP SET NUM VALUE (*rangeObj* : Object ; *numberValue* : Number { ; *formatPattern* : Text })

引数	タイプ		説明
<i>rangeObj</i>	Object	->	レンジオブジェクト
<i>numberValue</i>	Number	->	設定する数値
<i>formatPattern</i>	Text	->	値のフォーマット

説明

`VP SET NUM VALUE` コマンドは、指定のセルレンジに数値を割り当てます。

rangeObj には、値を割り当てたいセルのレンジ (たとえば [VP Cell](#) あるいは [VP Column](#) で作成されたレンジ) を渡します。*rangeObj* 引数に複数のセルが含まれる場合、指定された値はそれぞれのセルに対して繰り返し割り当てられます。

newValue 引数に、*rangeObj* 引数のレンジに割り当てたい数値を指定します。

任意の *formatPattern* 引数は、*newValue* に対する [パターン](#) を定義します。

例題

```
// セルに2という値を設定します  
VP SET NUM VALUE(VP Cell("ViewProArea";3;2);2)  
  
// セルの値を設定し、フォーマットをドル表記に設定します  
VP SET NUM VALUE(VP Cell("ViewProArea";3;2);12.356;"_($* #,##0.00_)")
```

参照

[Cell format](#)

[VP SET VALUE](#)

VP SET PRINT INFO

VP SET PRINT INFO (*vpAreaName* : Text ; *printInfo* : Object { ; *sheet* : Integer })

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro エリア名
<i>printInfo</i>	Object	->	印刷属性を格納するオブジェクト
<i>sheet</i>	Integer	->	シートのインデックス (省略した場合はカレントシート)

説明

VP SET PRINT INFO コマンドは、*vpAreaName* 引数で指定したエリアを印刷する際に使用する属性を定義します。

vpAreaName には、印刷する 4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

printInfo には、様々な印刷属性の定義を格納したオブジェクトを渡します。利用可能な属性の一覧については、[4D View Pro 印刷属性](#) を参照してください。

任意の *sheet* 引数として、シートのインデックス (0 起点) を渡すことで、印刷するスプレッドシートを指定することができます。省略された場合はデフォルトでカレントスプレッドシートが使用されます。以下の定数を使用することでカレントのスプレッドシートを明示的に選択することができます:

- `vk current sheet`

例題

以下のコードを実行すると、4D View Pro エリアを PDF ドキュメントに出力します:

```

var $printInfo : Object

// 印刷属性オブジェクトを宣言します
$printInfo:=New object

// 印刷属性を定義します
$printInfo.headerCenter:="&BS.H.I.E.L.D. &A Sales Per Region"
$printInfo.firstPageNumber:=1
$printInfo.footerRight:="page &P of &N"
$printInfo.orientation:=vk print page orientation landscape
$printInfo.centering:=vk print centering horizontal
$printInfo.columnStart:=0
$printInfo.columnEnd:=8
$printInfo.rowStart:=0
$printInfo.rowEnd:=24

$printInfo.showGridLine:=True

// 会社のロゴを追加します
$printInfo.headerLeftImage:=logo.png
$printInfo.headerLeft:="&G"

$printInfo.showRowHeader:=vk print visibility hide
$printInfo.showColumnHeader:=vk print visibility hide
$printInfo.fitPagesWide:=1
$printInfo.fitPagesTall:=1

// 印刷情報を設定します
VP SET PRINT INFO ("ViewProArea";$printInfo)

// PDF を書き出します
VP EXPORT DOCUMENT("ViewProArea";"Sales2018.pdf";New object("formula");Formula(ALERT("PDF ready!")))

```

出力されたPDF:



S.H.I.E.L.D. 2018 Sales Per Region

OrderDate	Region	Rep	Item	Units	UnitCost	Total
06-Jan-18	East	Stark	Pencil	95	\$1.99	\$189.05
23-Jan-18	Central	Rogers	Binder	50	\$19.99	\$999.50
09-Feb-18	Central	Banner	Pencil	36	\$4.99	\$179.64
26-Feb-18	Central	Romanoff	Pen	27	\$19.99	\$539.73
15-Mar-18	West	Barton	Pencil	56	\$2.99	\$167.44
01-Apr-18	East	Coulson	Binder	60	\$4.99	\$299.40
18-Apr-18	Central	Fury	Pencil	75	\$1.99	\$149.25
05-May-18	Central	Potts	Pencil	90	\$4.99	\$449.10
22-May-18	West	Jarvis	Pencil	32	\$1.99	\$63.68
08-Jun-18	East	Loki	Binder	60	\$8.99	\$539.40
25-Jun-18	Central	Odin	Pencil	90	\$4.99	\$449.10
06-Jul-18	East	Stark	Pencil	95	\$1.99	\$189.05
23-Jul-18	Central	Rogers	Binder	50	\$19.99	\$999.50
09-Aug-18	Central	Banner	Pencil	36	\$4.99	\$179.64
26-Aug-18	Central	Romanoff	Pen	27	\$19.99	\$539.73
15-Sep-18	West	Barton	Pencil	56	\$2.99	\$167.44
01-Oct-18	East	Coulson	Binder	60	\$4.99	\$299.40
18-Oct-18	Central	Fury	Pencil	75	\$1.99	\$149.25
05-Nov-18	Central	Potts	Pencil	90	\$4.99	\$449.10
22-Nov-18	West	Jarvis	Pencil	32	\$1.99	\$63.68
08-Dec-18	East	Loki	Binder	60	\$8.99	\$539.40
15-Dec-18	Central	Odin	Pencil	90	\$4.99	\$449.10
				1,342	\$155.78	\$8,050.58

page 1 of 1

参照

[4D View Pro print attributes](#)[VP Convert to picture](#)[VP Get print info](#)[VP PRINT](#)

VP SET ROW ATTRIBUTES

VP SET ROW ATTRIBUTES (*rangeObj* : Object ; *propertyObj* : Object)

引数	タイプ		説明
<i>rangeObj</i>	Object	->	行レンジ
<i>propertyObj</i>	Object	->	行のプロパティを格納したオブジェクト

説明

VP SET ROW ATTRIBUTES コマンドは、*rangeObj* で指定した行に対して、*propertyObj* に定義された属性を適用します。*rangeObj* 引数には、レンジオブジェクトを渡します。レンジにカラムと行の両方が格納されている場合、属性はカラムに対してのみ適用されます。*propertyObj* 引数は、*rangeObj* 引数のレンジ内の行に対して適用する属性を指定します。指定できる属性は以下の通りです：

プロパティ	タイプ	説明
height	number	行の高さ (ピクセル単位)
pageBreak	boolean	レンジ内の先頭行の前に改ページを挿入する場合には true、それ以外は false
visible	boolean	行が表示状態であれば true、それ以外は false
resizable	boolean	行がリサイズ可能であれば true、それ以外は false
headers	text	行ヘッダーのテキスト

例題

2番目の行の高さを変更して、ヘッダーを設定します:

```
var $row; $properties : Object

$row:=VP Row("ViewProArea";1)
$properties:=New object("height";75;"header";"June")

VP SET ROW ATTRIBUTES($row;$properties)
```

	A	B	C	D	E	F	G	H	I
1	The	quick	brown	fox	jumped	over	the	lazy	dog
	June								

参照

[VP Get row attributes](#)
[VP get column attributes](#)
[VP SET ROW ATTRIBUTES](#)

VP SET ROW COUNT

VP SET ROW COUNT (*vpAreaName* : Text ; *rowCount* : Integer { ; *sheet* : Integer })

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro フォームオブジェクト名
<i>rowCount</i>	Integer	->	行数
<i>sheet</i>	Integer	->	シートのインデックス (省略した場合はカレントシート)

説明

`VP SET ROW COUNT` コマンドは、*vpAreaName* 引数内にある行の総数を定義します。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

rowCount には、行の総数を渡します。*rowCount* 引数は 0 より大きい値でなければなりません。

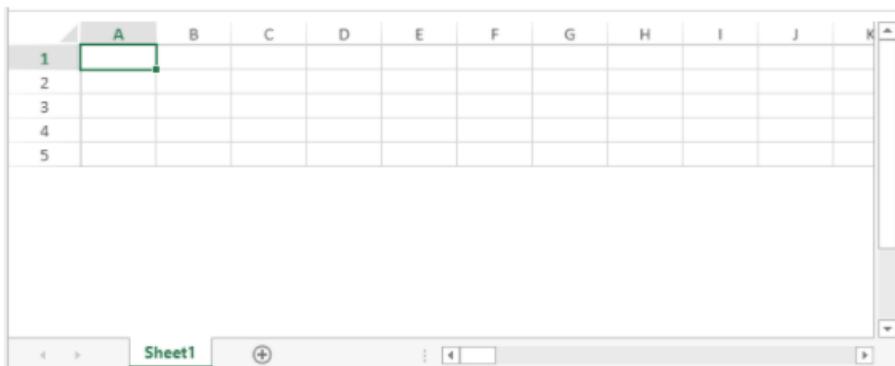
任意の *sheet* 引数として、シートのインデックス (0 起点) を渡すことで、*rowCount* が適用されるスプレッドシートを指定することができます。省略された場合はデフォルトでカレントスプレッドシートが使用されます。以下の定数を使用することでカレントのスプレッドシートを明示的に選択することができます:

- `vk current sheet`

例題

以下のコードは 4D View Pro エリア内に 5つの行を定義します:

```
VP SET ROW COUNT("ViewProArea";5)
```



参照

[VP Get column count](#)

[VP get row-count](#)

[VP SET COLUMN COUNT](#)

VP SET SELECTION

VP SET SELECTION (*rangeObj* : Object)

引数	タイプ		説明
<i>rangeObj</i>	Object	->	複数セルのレンジオブジェクト

説明

VP SET SELECTION コマンドは、指定のセルレンジを選択し、その先頭セルをアクティブセルに設定します。

rangeObj には、カレントセレクションとして定義するセルのレンジオブジェクトを渡します。

例題

```
$currentSelection:=VP Combine ranges(VP Cells("myVPArea";3;2;1;6);VP Cells("myVPArea";5;7;1;7))
VP SET SELECTION($currentSelection)
```

	A	B	C	D	E	F	G	
1								
2								
3	Month	Area	Mr. Smith	Ms. Johnson	Ms. Williams	Mr. Jones	M	
4	January	South	1898	1857	1651	1448		
5		East	4859	4857	2548	4876		
6		North	2458	1524	6150	4987		
7		West	5787	1580	3975	4878		
8		Total	15002	9818	14324	16189		
9	February	South	6668	4374	17495	9999		
10		East	5955	1677	7944	9400		
11		North	1000	6722	2195	2777		
12		West	6896	8355	7195	2058		
13		Total	20519	21128	34829	24234		
14	March	South	2577	2000	6185	2704		
15		East	4859	4857	2548	4876		
16		North	2458	1524	6150	4987		
17		West	5787	1580	3975	4878		
		Total	15621	9961	18852	17115		

参照

[VP Get active cell](#)

[VP Get selection](#)

[VP RESET SELECTION](#)

[VP SET ACTIVE CELL](#)

[VP ADD SELECTION](#)

[VP SHOW CELL](#)

VP SET SHEET COUNT

VP SET SHEET COUNT (vpAreaName : Text ; number : Integer)

引数	タイプ		説明
vpAreaName	Text	->	4D View Pro フォームオブジェクト名
number	Integer	->	シートの数

説明

VP SET SHEET COUNT コマンドは、vpAreaName 引数で指定したView Pro エリア内のシートの数を設定します。

number 引数には、コマンド実行後にドキュメントが格納するシート数を指定する数値を渡します。

警告: このコマンドは、現在のシート数より少ない数字を渡した場合にはシートを削除します。たとえば、ドキュメント内にシートが 5つあり、このコマンドでシートを 3つに設定した場合には、シート4 と 5 は削除されます。

例題

ドキュメントには現在シートが 1つあります：



シート数を 3つに設定します：

```
VP SET SHEET COUNT("ViewProArea";3)
```



参照

[VP Get sheet count](#)

VP SET SHEET NAME

VP SET SHEET NAME (*vpAreaName* : Text ; *name* : Text {; *index*: Integer})

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro フォームオブジェクト名
<i>name</i>	Text	->	シートの新しい名称
<i>index</i>	Integer	->	名称変更するシートのインデックス

説明

VP SET SHEET NAME コマンドは、*vpAreaName* 引数で指定した View Pro エリア内にロードされているドキュメント内のシート名を変更します。

vpAreaName には、4D View Pro エリアの名前を渡します。

name 引数として、シートの新しい名前を渡します。

index 引数には、名称変更するシートのインデックスを渡します。

インデックスは 0 起点です。

index が省略された場合、コマンドはカレントシートを名称変更します。

新しい名前には、次の文字を含めることはできません: *, :, [,], ?, \, /

このコマンドは、以下の場合には何もしません:

- 新しい名前に禁止文字が含まれている
- 新しい名前が空の文字列である
- 新しい名前が既に存在している
- index* に渡したインデックスが存在しない

例題

3つ目のシートの名前を "Total first quarter" に変更します:

```
VP SET SHEET NAME("ViewProArea";"Total first quarter";2)
```



VP SET SHEET OPTIONS

VP SET SHEET OPTIONS (*vpAreaName* : Text; *sheetOptions* : Object { ; *sheet* : Integer})

引数	タイプ		説明
vpAreaName	Object	->	4D View Pro エリア名
sheetOptions	Object	->	設定するシートオプション
sheet	Object	->	シートのインデックス (省略した場合はカレントシート)

説明

`VP SET SHEET OPTIONS` コマンドは、`vpAreaName` 引数で名前を指定した View Pro エリアの様々なシートオプションを設定します。

`vpAreaName` には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

`sheetOptions` には、設定するオプションの定義を格納したオブジェクトを渡します。利用可能なオプション一覧については、[シートオプション](#) を参照ください。

任意の `sheet` 引数として、シートのインデックス (0 起点) を渡すことで、スプレッドシートを指定することができます。省略された場合はデフォルトでカレントスプレッドシートが使用されます。以下の定数を使用することでカレントのスプレッドシートを明示的に選択することができます:

- `vk current sheet`

例題 1

C5:D10 のレンジ以外のセルをすべて保護します:

```
// カレントシートでセルの保護を有効化します
var $options : Object

$options:=New object
$options.isProtected:=True
VP SET SHEET OPTIONS("ViewProArea";$options)

// C5:D10 を 'unlocked' に設定します
VP SET CELL STYLE(VP Cells("ViewProArea";2;4;2;6);New object("locked";False))
```

例題 2

ドキュメントを保護しつつ、ユーザーが行とカラムをリサイズできるようにします:

```
var $options : Object

$options:=New object
// 保護を有効化します
$options.isProtected:=True
$options.protectionOptions:=New object
// ユーザーに行のリサイズを許可します
$options.protectionOptions.allowResizeRows=True;
// ユーザーにカラムのリサイズを許可します
$options.protectionOptions.allowResizeColumns=True;

// カレントシートに上記の設定での保護を適用します
VP SET SHEET OPTIONS("ViewProArea";$options)
```

例題 3

シートのタブ、固定化された線、枠線、選択範囲の背景と選択範囲の境界線のカラーをカスタマイズします:

```

var $options : Object

$options:=New object
// Sheet1 のタブのカラーをカスタマイズします
$options.sheetTabColor:="Black"
$options.gridline:=New object("color";"Purple")
$options.selectionBackColor:="rgb(255,128,0,0.4)"
$options.selectionBorderColor:="Yellow"
$options.frozenlineColor:="Gold"

VP SET SHEET OPTIONS("ViewProArea";$options;0)

// Sheet2 のタブのカラーをカスタマイズします
$options.sheetTabColor:="red"

VP SET SHEET OPTIONS("ViewProArea";$options;1)

// Sheet3 のタブのカラーをカスタマイズします
$options.sheetTabColor:="blue"

VP SET SHEET OPTIONS("ViewProArea";$options;2)

```

結果:

A screenshot of a spreadsheet application interface. The top menu bar has 'File', 'Edit', 'View', 'Format', 'Insert', 'Tools', and 'Help'. Below the menu is a toolbar with icons for 'New', 'Open', 'Save', 'Print', 'Find', 'Replace', 'Copy', 'Cut', 'Paste', 'Delete', 'Format', 'Cell', 'Row', 'Column', 'Sort Ascending', 'Sort Descending', and 'Filter'. The main area shows a grid with columns labeled A through J and rows labeled 1 through 12. The first four rows (1-4) have green headers. Row 5 has a yellow header. Rows 6-12 have grey headers. The gridlines are purple. A yellow selection box highlights a range from F6 to H11. The tabs at the bottom are 'Sheet1' (black), 'Sheet2' (red), and 'Sheet3' (blue). The status bar at the bottom shows 'Sheet1' and '100%'. The bottom right corner has a small logo.

例題 4

枠線と、行ヘッダー/カラムヘッダーを非表示にします:

```

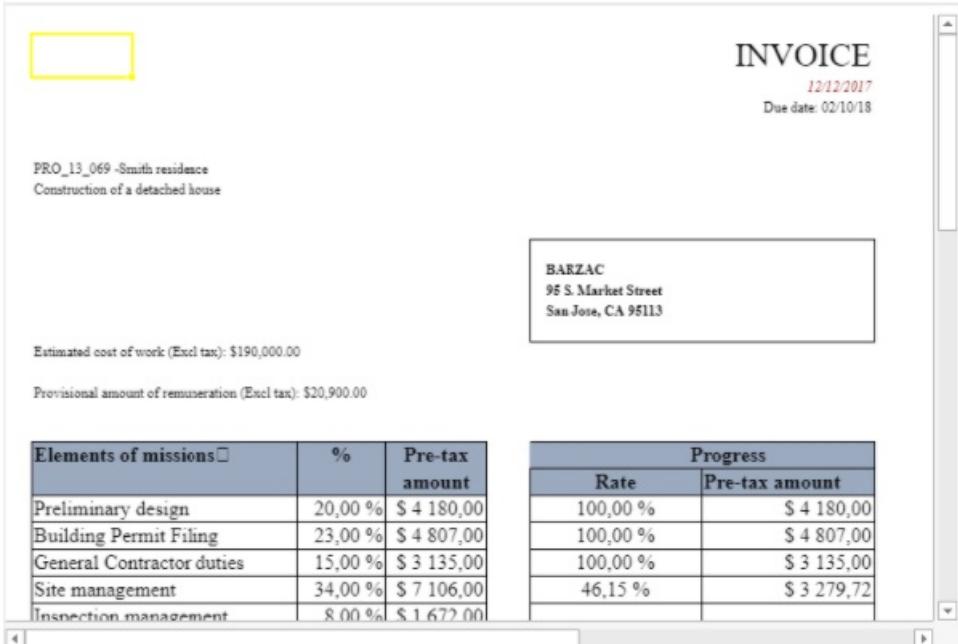
var $options : Object

$options:=New object
$options.gridline:=New object()
$options.gridline.showVerticalGridline:=False
$options.gridline.showHorizontalGridline:=False
$options.rowHeaderVisible:=False
$options.colHeaderVisible:=False

VP SET SHEET OPTIONS("ViewProArea";$options)

```

結果:



参照

[4D View Pro sheet options](#)

[VP Get sheet options](#)

VP SET SHOW PRINT LINES

VP SET SHOW PRINT LINES (*vpAreaName* : Text {; *visible* : Boolean}{; *index* : Integer})

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro フォームオブジェクト名
<i>visible</i>	Boolean	->	印刷線を表示する場合は true (デフォルト)、非表示の場合は false
<i>index</i>	Integer	->	シートのインデックス

説明

`VP SET SHOW PRINT LINES` コマンドは、スプレッドシート内で印刷プレビュー線を表示するかどうかを設定します。

vpAreaName には、4D View Pro エリアの名前を渡します。

visible には、印刷線を表示するには `True`、非表示にするには `False` を渡します。デフォルトでは `True` が渡されます。

index には、ターゲットシートのインデックスを渡します。*index* が省略された場合、コマンドはカレントシートに対して適用されます。

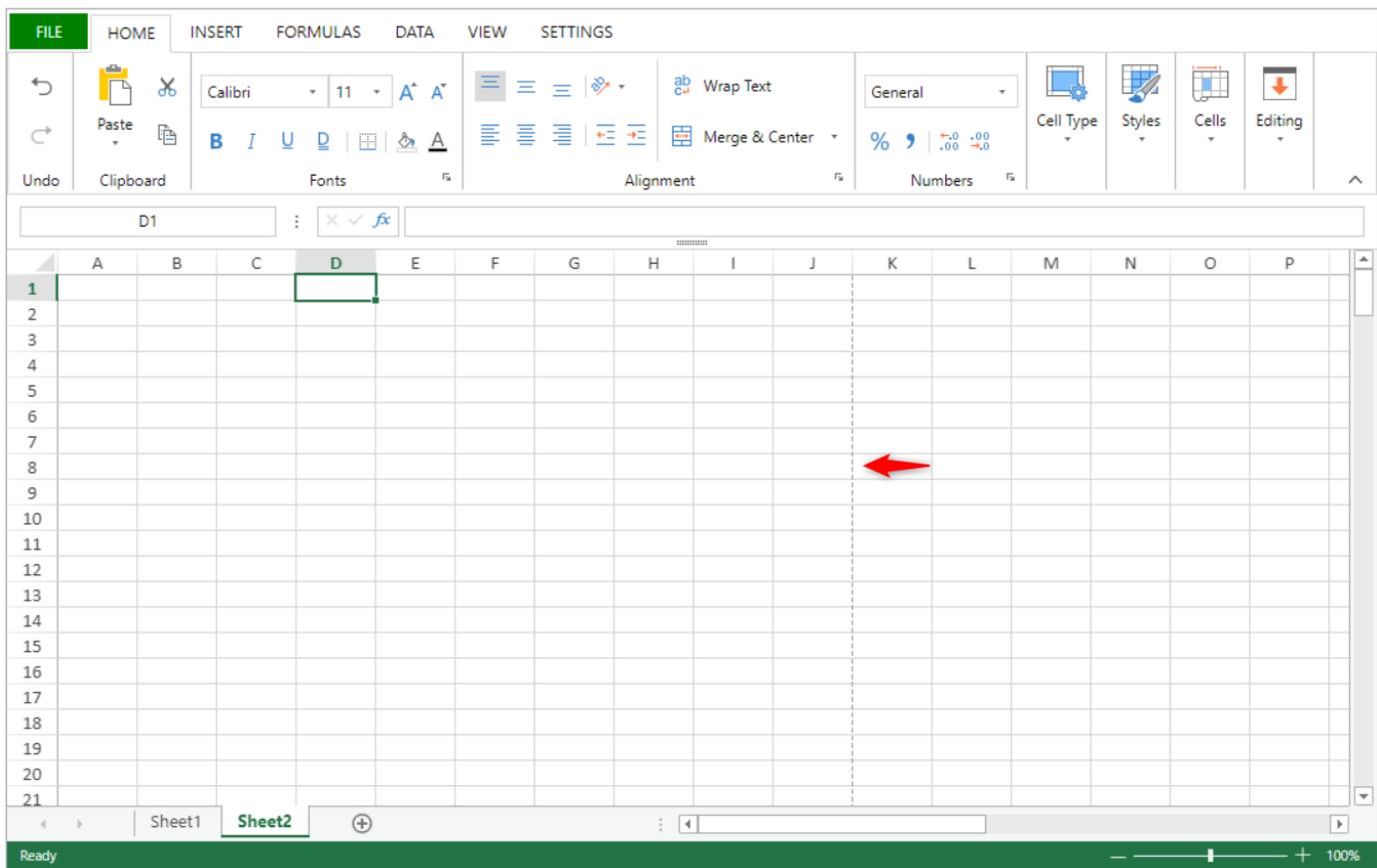
インデックスは 0 起点です。

スプレッドシートの印刷線の位置は、スプレッドシートの改ページの位置によって変化します。

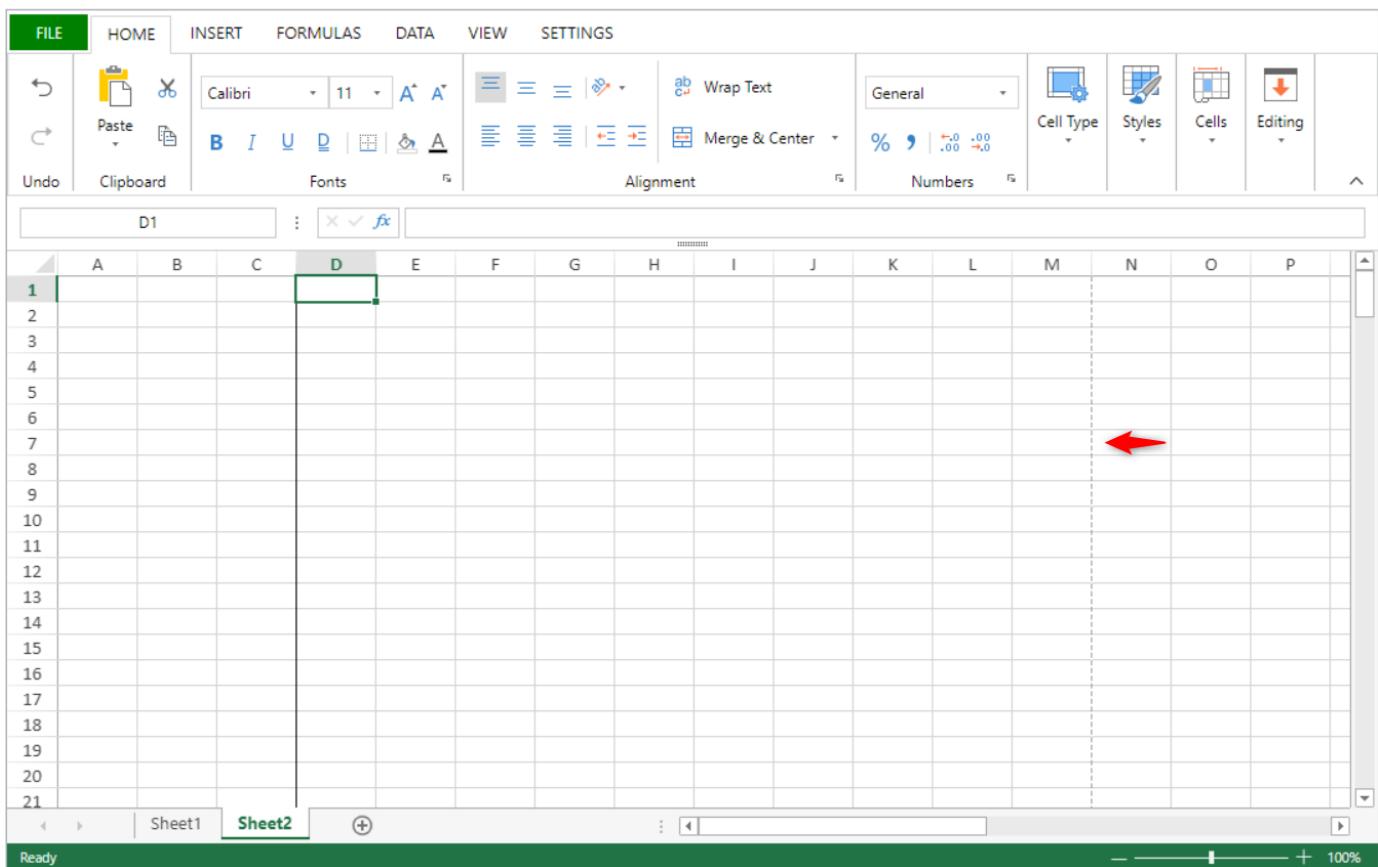
例題

以下のコードはドキュメントの 2番目のシートの印刷線を表示させます:

```
VP SET SHOW PRINT LINES("ViewProArea";True;1)
```



改ページがある場合:



参照

[4D Get show print lines](#)

VP SET TEXT VALUE

VP SET TEXT VALUE (*rangeObj* : Object ; *textValue* : Text { ; *formatPattern* : Text })

引数	タイプ		説明
rangeObj	Object	->	レンジオブジェクト
textValue	Text	->	設定するテキスト値
formatPattern	Text	->	値のフォーマット

説明

`VP SET TEXT VALUE` コマンドは、指定されたセルレンジにテキスト値を割り当てます。

rangeObj には、値を割り当てるセルのレンジ (たとえば `VP Cell` あるいは `VP Column` で作成されたレンジ) を渡します。 *rangeObj* 引数に複数のセルが含まれる場合、指定された値はそれぞれのセルに対して繰り返し割り当てられます。

textValue 引数に、*rangeObj* 引数のレンジに割り当てるテキスト値を指定します。

任意の *formatPattern* 引数は、*textValue* に対する [パターン](#) を定義します。

例題

```
VP SET TEXT VALUE(VP Cell("ViewProArea";3;2);"Test 4D View Pro")
```

参照

[Cell Format](#)

[VP SET VALUE](#)

VP SET TIME VALUE

`VP SET TIME VALUE (rangeObj : Object ; timeValue : Text { ; formatPattern : Text })`

引数	タイプ		説明
rangeObj	Object	->	レンジオブジェクト
timeValue	Text	->	設定する時間値
formatPattern	Text	->	値のフォーマット

説明

`VP SET TIME VALUE` コマンドは、指定されたセルレンジに時間値を割り当てます。

rangeObj には、値を割り当てるセルのレンジ (たとえば `VP Cell` あるいは `VP Column` で作成されたレンジ) を渡します。 *rangeObj* 引数に複数のセルが含まれる場合、指定された値はそれぞれのセルに対して繰り返し割り当てられます。

timeValue 引数に、*rangeObj* 引数のレンジに割り当てる時間 (秒単位) を指定します。

任意の *formatPattern* 引数は、*timeValue* に対する [パターン](#) を定義します。

例題

```
// セルの値を現在の時間に設定します
VP SET TIME VALUE(VP Cell("ViewProArea";5;2);Current time)
```

```
// セルの値を、指定されたフォーマットの特定の時間に設定します
VP SET TIME VALUE(VP Cell("ViewProArea";5;2);?12:15:06?vk pattern long time)
```

参照

[Cell Format](#)

[VP SET DATE TIME VALUE](#)
[VP SET VALUE](#)

VP SET VALUE

VP SET VALUE (*rangeObj* : Object ; *valueObj* : Object)

引数	タイプ		説明
<i>rangeObj</i>	Object	->	レンジオブジェクト
<i>valueObj</i>	Object	->	セルの値とフォーマットオプション

説明

VP SET VALUE コマンドは、指定されたセルレンジに値を割り当てます。

このコマンドを使用すると、汎用的なコードで *rangeObj* のレンジに様々な型の値とそのフォーマットを設定できます。それに対して [VP SET TEXT VALUE](#) や [VP SET NUM VALUE](#) などの他のコマンドは、設定する値の型が限定されています。

rangeObj には、値を割り当てるセルのレンジ (たとえば [VP Cell](#) あるいは [VP Column](#) で作成されたレンジ) を渡します。*rangeObj* 引数に複数のセルが含まれる場合、指定された値はそれぞれのセルに対して繰り返し割り当てられます。

valueObj 引数は、*rangeObj* のレンジに対して割り当てる値と [フォーマット](#) のプロパティを格納しているオブジェクトです。このオブジェクトには以下のプロパティを含めることができます:

プロパティ	タイプ	説明
<i>value</i>	Integer, Real, Boolean, Text, Date, Null	<i>rangeObj</i> のレンジに対して割り当てる値 (時間型を除く)。セルの中身を消去するためには Null を渡します。
<i>time</i>	Real	<i>rangeObj</i> のレンジに対して割り当てる時間 (秒単位)
<i>format</i>	Text	値や日時に対するパターン 値や日時に対するパターン パターンおよびフォーマット文字についての情報については、 セルフォーマット の章を参照してください。

例題

```

// セルの値を False に設定します
VP SET VALUE(VP Cell("ViewProArea";3;2);New object("value";False))

// セルの値を 2 に設定します
VP SET VALUE(VP Cell("ViewProArea";3;2);New object("value";2))

// セルの値を $125,571.35 に設定します
VP SET VALUE(VP Cell("ViewProArea";3;2);New object("value";125571.35;"format";"_(#$#,##0.00_)"))

// セルの値を Hello World!
に設定します
VP SET VALUE(VP Cell("ViewProArea";3;2);New object("value";"Hello World!"))

// セルの値を現在の日付に設定します
VP SET VALUE(VP Cell("ViewProArea";4;2);New object("value";Current date))

// セルの値を現在の時間に設定します
VP SET VALUE(VP Cell("ViewProArea";5;2);New object("time";Current hour))

// セルの値を特定の日付と時間に設定します
VP SET VALUE(VP Cell("ViewProArea";3;9);New object("value";!2024-12-18!);"time";?14:30:10?;"format";vk p)

// セルの中身を消去します
VP SET VALUE(VP Cell("ViewProArea";3;9);New object("value";Null))

```

参照

[Cell Format](#)
[VP Get values](#)
[VP SET VALUE](#)
[VP SET BOOLEAN VALUE](#)
[VP SET DATE TIME VALUE](#)
[VP SET FIELD](#)
[VP SET FORMULA](#)
[VP SET NUM VALUE](#)
[VP SET TEXT VALUE](#)
[VP SET TIME VALUE](#)

VP SET VALUES

VP SET VALUES (*rangeObj* : Object ; *valuesCol* : Collection)

引数	タイプ		説明
<i>rangeObj</i>	Object	->	レンジオブジェクト
<i>valuesCol</i>	Collection	->	値のコレクション

説明

VP SET VALUES コマンドは、指定のセルレンジから開始して値のコレクションを割り当てていきます。

rangeObj には、値を割り当てるセルのレンジ (たとえば [VP Cell](#) あるいは [VP Column](#) で作成されたレンジ) を渡します。 *rangeObj* 引数に複数のセルが含まれる場合、指定された値はそれぞれのセルに対して繰り返し割り当てられます。

- *rangeObj* のレンジが複数レンジを指定している場合、最初のレンジの先頭セルのみが使用されます。
- *rangeObj* のレンジが複数レンジを指定している場合、最初のレンジの先頭セルのみが使用されます。

valuesCol 引数は 2次元構造のコレクションです:

- 第1レベルのコレクションは、値のサブコレクションを格納しています。それぞれのサブコレクションは行を定義します。行をスキップするには空のコレクションを渡します。
- それぞれのサブコレクションは行におけるセルの値を定義します。値は整数、実数、布尔、テキスト、日付、Null、オブジェクトのいずれかです。値がオブジェクトの場合、以下のプロパティを持つことができます：

プロパティ	タイプ	説明
value	Integer, Real, Boolean, Text, Date, Null	セルの値 (時間部分を除く)
time	Real	時間値 (秒単位)

例題

```
$param:=New collection
$param.push(New collection(1;2;3;False)) // 1行目用に 4つの値を設定します
$param.push(New collection) // 2行目は空行です
$param.push(New collection(4;5;Null;"hello";"world")) // 3行目用に 5つの値を設定します
$param.push(New collection(6;7;8;9)) // 4行目用に 4つの値を設定します
$param.push(New collection(Null;New object("value";Current date;"time";42))) // 5行目用に 1つの値を設定します

VP SET VALUES(VP Cell("ViewProArea";2;1);$param)
```

	A	B	C	D	E	F	G
1							
2			1	2	3	FALSE	
3							
4			4	5	hello	world	
5			6	7	8	9	
6			29/05/2019 0:00:42				
7							

参照

[VP Get formulas](#)
[VP Get value](#)
[VP Get Values](#)
[VP SET FORMULAS](#)
[VP SET VALUE](#)

VP SET WORKBOOK OPTIONS

VP SET WORKBOOK OPTIONS (*vpAreaName* : Text ; *optionObj* : Object)

引数	タイプ		説明
<i>vpAreaName</i>	Text	->	4D View Pro フォームオブジェクト名
<i>optionObj</i>	Object	->	設定するワークブックオプションを格納したオブジェクト

説明

VP SET WORKBOOK OPTIONS コマンドは、*vpAreaName* 引数で指定した View Pro エリアのワークブックオプションを設定します。

vpAreaName には、4D View Pro エリアの名前を渡します。

*optionObj*には、*vpAreaName* のエリアに対して適用するワークブックオプションを渡します。

optionObj 引数が空の場合、このコマンドは何もしません。

変更されたワークブックオプションはドキュメントとともに保存されます。

次の表は、利用可能なワークブックオプションの一覧です:

プロパティ	タイプ	説明															
allowUserDragMerge	boolean	ドラッグ & マージオプションを許可します (複数セルを選択し、選択をドラッグしてセルを結合します)															
allowAutoCreateHyperlink	boolean	スプレッドシート内でハイパーアンクの自動生成を有効にします。															
allowContextMenu	boolean	ビルトインのコンテキストメニューの使用を許可します。															
allowCopyPasteExcelStyle	boolean	スプレッドシートのスタイルを Excel にコピー & ペーストすることを許可します (逆も可)。															
allowDynamicArray	boolean	ワークシート内で動的配列を有効にします。															
allowExtendPasteRange	boolean	貼り付けデータが貼り付け先の範囲に収まりきらない場合に、貼り付け先の範囲を拡張します。															
allowSheetReorder	boolean	シートの順序変更を許可します。															
allowUndo	boolean	編集を元に戻す操作を許可します。															
allowUserDeselect	boolean	選択範囲から特定のセルを除外することを許可します。															
allowUserDragDrop	boolean	レンジデータのドラッグ & ドロップを許可します。															
allowUserDragFill	boolean	ドラッグ & フィルを許可します。															
allowUserEditFormula	boolean	セルへのフォーミュラの入力を許可します。															
allowUserResize	boolean	カラムと行のリサイズを許可します。															
allowUserZoom	boolean	ズームを許可します (Ctrl + マウスホイール)。															
autoFitType	number	セル内やヘッダー内に収まるよう、内容をフォーマットします。使用可能な値: <table border="1"><thead><tr><th>定数</th><th>値</th><th>説明</th></tr></thead><tbody><tr><td>vk auto fit type cell</td><td>0</td><td>内容をセル内に収めます。</td></tr><tr><td>vk auto fit type cell with header</td><td>1</td><td>内容をセル内・ヘッダー内に収めます。</td></tr></tbody></table>	定数	値	説明	vk auto fit type cell	0	内容をセル内に収めます。	vk auto fit type cell with header	1	内容をセル内・ヘッダー内に収めます。						
定数	値	説明															
vk auto fit type cell	0	内容をセル内に収めます。															
vk auto fit type cell with header	1	内容をセル内・ヘッダー内に収めます。															
backColor	string	エリアの背景色を表すカラー文字列 (例: "red"、"#FFFF00"、"rgb(255,0,0)"、"Accent 5")。backgroundImage を設定している場合、背景色は非表示になります。															
backgroundImage	string / picture / file	エリアの背景画像。															
backgroundImageLayout	number	背景画像のレイアウト。使用可能な値: <table border="1"><thead><tr><th>定数</th><th>値</th><th>説明</th></tr></thead><tbody><tr><td>vk image layout center</td><td>1</td><td>エリアの中央に表示。</td></tr><tr><td>vk image layout none</td><td>3</td><td>エリアの左上に元のサイズで表示。</td></tr><tr><td>vk image layout stretch</td><td>0</td><td>エリアを埋めるように拡大表示。</td></tr><tr><td>vk image layout zoom</td><td>2</td><td>アスペクト比を維持して表示。</td></tr></tbody></table>	定数	値	説明	vk image layout center	1	エリアの中央に表示。	vk image layout none	3	エリアの左上に元のサイズで表示。	vk image layout stretch	0	エリアを埋めるように拡大表示。	vk image layout zoom	2	アスペクト比を維持して表示。
定数	値	説明															
vk image layout center	1	エリアの中央に表示。															
vk image layout none	3	エリアの左上に元のサイズで表示。															
vk image layout stretch	0	エリアを埋めるように拡大表示。															
vk image layout zoom	2	アスペクト比を維持して表示。															
calcOnDemand	boolean	要求されたときのみフォーミュラを計算します。															
columnResizeMode	number	カラムのリサイズモード。使用可能な値:															

プロパティ	タイプ	説明	値	説明
		vk resize mode normal	0	通常のリサイズモード (残りのカラムに影響します)
		vk resize mode split	1	split モード (残りのカラムに影響しません)
copyPasteHeaderOptions	number	データのコピー/ペースト時に含めるヘッダーについて指定します。使用可能な値:		
		定数	値	説明
		vk copy paste header options all headers	3	データのコピー時: 選択ヘッダーを含めます。
		vk copy paste header options column headers	2	データのコピー時: 選択されたカラムヘッダーを含めます。
		vk copy paste header options no headers	0	データのコピー時: ヘッダーを含めません。データのペースト時: ヘッダーを上書きしません。
		vk copy paste header options row headers	1	データのコピー時: 選択された行ヘッダーを含めます。
customList	collection	ドラッグ & フィルをカスタマイズするためのリストです。ファイルの際には、このリストに合致する値が入力されます。各コレクション要素は、文字列のコレクションです。 GrapeCity の Webサイト 参照。		
cutCopyIndicatorBorderColor	string	ユーザーが選択をカットまたはコピーしたときの領域の境界色。		
cutCopyIndicatorVisible	boolean	コピーまたはカットされた際の領域を表示します。		
defaultDragFillType	number	デフォルトのドラッグ & フィルタイプ。使用可能な値 :		
		定数	値	説明
		vk auto fill type auto	5	自動でセルをフィルします。
		vk auto fill type clear values	4	セルの値をクリアします。
		vk auto fill type copycells	0	値・フォーマット・フォーミュラを含むすべてのデータオブジェクトでセルをフィルします。
		vk auto fill type fill formatting only	2	フォーマットのみでセルをフィルします。
		vk auto fill type fill series	1	連続データでフィルします。
		vk auto fill type fill without formatting	3	値のみでセルをフィルします (フォーマットは除外)。
enableAccessibility	boolean	スプレッドシートにおけるアクセシビリティのサポートを有効にします。		
enableFormulaTextbox	boolean	フォーミュラテキストボックスを有効化します。		
grayAreaBackColor	string	グレー領域の背景色を表すカラー文字列 (例: "red"、"#FFFF00"、"rgb(255,0,0)"、"Accent 5")。		

highlightInvalidData プロパティ	boolean タイプ	無効なデータをハイライト表示します。									
iterativeCalculation	boolean	反復計算を有効にします。 Grapecity の Webサイト 参照。									
iterativeCalculationMaximumChange	numeric	2つの計算値の最大差。									
iterativeCalculationMaximumIterations	numeric	フォーミュラが反復計算される最大回数。									
newTabVisible	boolean	新規シートを挿入するための特別なタブを表示します。									
numbersFitMode	number	<p>日付/数値データがカラム幅を超える場合の表示モード。使用可能な値:</p> <table border="1"> <thead> <tr> <th>定数</th> <th>値</th> <th>説明</th> </tr> </thead> <tbody> <tr> <td>vk numbers fit mode mask</td> <td>0</td> <td>データの中身を "###" で置き換え、Tip を表示します。</td> </tr> <tr> <td>vk numbers fit mode overflow</td> <td>1</td> <td>データの中身を文字列として表示します。隣のセルが空であれば、はみ出して表示します。</td> </tr> </tbody> </table>	定数	値	説明	vk numbers fit mode mask	0	データの中身を "###" で置き換え、Tip を表示します。	vk numbers fit mode overflow	1	データの中身を文字列として表示します。隣のセルが空であれば、はみ出して表示します。
定数	値	説明									
vk numbers fit mode mask	0	データの中身を "###" で置き換え、Tip を表示します。									
vk numbers fit mode overflow	1	データの中身を文字列として表示します。隣のセルが空であれば、はみ出して表示します。									
pasteSkipInvisibleRange	boolean	<p>非表示のレンジへの貼り付けについて指定します。</p> <ul style="list-style-type: none"> False (デフォルト): データを貼り付けます。 True: 非表示のレンジはスキップします。 <p>非表示のレンジについての詳細は Grapecity' のドキュメント を参照ください。</p>									
referenceStyle	number	<p>セルフォーミュラにおける、セルやレンジ参照のスタイル。使用可能な値:</p> <table border="1"> <thead> <tr> <th>定数</th> <th>値</th> <th>説明</th> </tr> </thead> <tbody> <tr> <td>vk reference style A1</td> <td>0</td> <td>A1 スタイルを使用します。</td> </tr> <tr> <td>vk reference style R1C1</td> <td>1</td> <td>R1C1 スタイルを使用します。</td> </tr> </tbody> </table>	定数	値	説明	vk reference style A1	0	A1 スタイルを使用します。	vk reference style R1C1	1	R1C1 スタイルを使用します。
定数	値	説明									
vk reference style A1	0	A1 スタイルを使用します。									
vk reference style R1C1	1	R1C1 スタイルを使用します。									
resizeZeroIndicator	number	<p>行やカラムのサイズが 0 に変更されたときの描画ポリシー。使用可能な値:</p> <table border="1"> <thead> <tr> <th>定数</th> <th>値</th> <th>説明</th> </tr> </thead> <tbody> <tr> <td>vk resize zero indicator default</td> <td>0</td> <td>行やカラムのサイズが 0 に変更されたときに、現在の描画ポリシーを使用します。</td> </tr> <tr> <td>vk resize zero indicator enhanced</td> <td>1</td> <td>行やカラムのサイズが 0 に変更されたときに、2本の短い線を描画します。</td> </tr> </tbody> </table>	定数	値	説明	vk resize zero indicator default	0	行やカラムのサイズが 0 に変更されたときに、現在の描画ポリシーを使用します。	vk resize zero indicator enhanced	1	行やカラムのサイズが 0 に変更されたときに、2本の短い線を描画します。
定数	値	説明									
vk resize zero indicator default	0	行やカラムのサイズが 0 に変更されたときに、現在の描画ポリシーを使用します。									
vk resize zero indicator enhanced	1	行やカラムのサイズが 0 に変更されたときに、2本の短い線を描画します。									
rowResizeMode	number	行のリサイズモード。使用可能な値は columnResizeMode と同じです。									
scrollbarAppearance	number	<p>スクロールバーの見た目。使用可能な値:</p> <table border="1"> <thead> <tr> <th>定数</th> <th>値</th> <th>説明</th> </tr> </thead> <tbody> <tr> <td>vk scrollbar appearance mobile</td> <td>1</td> <td>モバイル風のスクロールバー。</td> </tr> <tr> <td>vk scrollbar appearance skin (デフォルト)</td> <td>0</td> <td>Excel風のクラシックなスクロールバー。</td> </tr> </tbody> </table>	定数	値	説明	vk scrollbar appearance mobile	1	モバイル風のスクロールバー。	vk scrollbar appearance skin (デフォルト)	0	Excel風のクラシックなスクロールバー。
定数	値	説明									
vk scrollbar appearance mobile	1	モバイル風のスクロールバー。									
vk scrollbar appearance skin (デフォルト)	0	Excel風のクラシックなスクロールバー。									
scrollbarMaxAlign	boolean	スクロールバーをアクティブシートの最後の行およびカラムに揃えます。									
scrollbarShowMax	boolean	シートのカラムと行の総数に基づいてスクロールバーを表示します。									
scrollByPixel	boolean	ピクセル単位のスクロールを有効にします。									
scrollIgnoreHidden	boolean	スクロールバーは非表示の行やカラムを無視します。									
scrollPixel	integer	scrollByPixel が true の場合、スクロール毎のピクセル数を指定します。最終的にスクロールするピクセル数は scrolling delta (スクロールの相									

プロパティ	タイプ	対変化値) * scrollPixel によって算出されます。例: scrolling delta 説明が3、scrollPixel が5 の場合、最終的なスクロールピクセル数は 15 です。															
showDragDropTip	boolean	ドラッグ & ドロップの Tip を表示します。															
showDragFillSmartTag	boolean	ドラッグ & フィルダイアログを表示します。															
showDragFillTip	boolean	ドラッグ & フィルの Tip を表示します。															
showHorizontalScrollbar	boolean	横スクロールバーを表示します。															
showResizeTip	number	<p>リサイズ Tip の表示を指定します。使用可能な値:</p> <table border="1"> <thead> <tr> <th>定数</th> <th>値</th> <th>説明</th> </tr> </thead> <tbody> <tr> <td>vk show resize tip both</td> <td>3</td> <td>縦と横のリサイズ Tip が表示されます。</td> </tr> <tr> <td>vk show resize tip column</td> <td>1</td> <td>横のリサイズ Tip のみ表示されます。</td> </tr> <tr> <td>vk show resize tip none</td> <td>0</td> <td>リサイズ Tip は表示されません。</td> </tr> <tr> <td>vk show resize tip row</td> <td>2</td> <td>縦のリサイズ Tip のみ表示されます。</td> </tr> </tbody> </table>	定数	値	説明	vk show resize tip both	3	縦と横のリサイズ Tip が表示されます。	vk show resize tip column	1	横のリサイズ Tip のみ表示されます。	vk show resize tip none	0	リサイズ Tip は表示されません。	vk show resize tip row	2	縦のリサイズ Tip のみ表示されます。
定数	値	説明															
vk show resize tip both	3	縦と横のリサイズ Tip が表示されます。															
vk show resize tip column	1	横のリサイズ Tip のみ表示されます。															
vk show resize tip none	0	リサイズ Tip は表示されません。															
vk show resize tip row	2	縦のリサイズ Tip のみ表示されます。															
showScrollTip	number	<p>スクロール Tip の表示を指定します。使用可能な値:</p> <table border="1"> <thead> <tr> <th>定数</th> <th>値</th> <th>説明</th> </tr> </thead> <tbody> <tr> <td>vk show scroll tip both</td> <td>3</td> <td>縦と横のスクロール Tip が表示されます。</td> </tr> <tr> <td>vk show scroll tip horizontal</td> <td>1</td> <td>横のスクロール Tip のみ表示されます。</td> </tr> <tr> <td>vk show scroll tip none</td> <td>スクロール Tip は表示されません。</td> <td></td> </tr> <tr> <td>vk show scroll tip vertical</td> <td>2</td> <td>縦のスクロール Tip のみ表示されます。</td> </tr> </tbody> </table>	定数	値	説明	vk show scroll tip both	3	縦と横のスクロール Tip が表示されます。	vk show scroll tip horizontal	1	横のスクロール Tip のみ表示されます。	vk show scroll tip none	スクロール Tip は表示されません。		vk show scroll tip vertical	2	縦のスクロール Tip のみ表示されます。
定数	値	説明															
vk show scroll tip both	3	縦と横のスクロール Tip が表示されます。															
vk show scroll tip horizontal	1	横のスクロール Tip のみ表示されます。															
vk show scroll tip none	スクロール Tip は表示されません。																
vk show scroll tip vertical	2	縦のスクロール Tip のみ表示されます。															
showVerticalScrollbar	boolean	縦スクロールバーを表示します。															
tabEditable	boolean	タブストリップの編集を有効にします。															
tabNavigationVisible	boolean	タブナビゲーションを表示します。															
tabStripPosition	number	<p>タブストリップの位置を指定します。使用可能な値:</p> <table border="1"> <thead> <tr> <th>定数</th> <th>値</th> <th>説明</th> </tr> </thead> <tbody> <tr> <td>vk tab strip position bottom</td> <td>0</td> <td>タブストリップはワークブックの下側に位置します。</td> </tr> <tr> <td>vk tab strip position left</td> <td>2</td> <td>タブストリップはワークブックの左側に位置します。</td> </tr> <tr> <td>vk tab strip position right</td> <td>3</td> <td>タブストリップはワークブックの右側に位置します。</td> </tr> <tr> <td>vk tab strip position top</td> <td>1</td> <td>タブストリップはワークブックの上側に位置します。</td> </tr> </tbody> </table>	定数	値	説明	vk tab strip position bottom	0	タブストリップはワークブックの下側に位置します。	vk tab strip position left	2	タブストリップはワークブックの左側に位置します。	vk tab strip position right	3	タブストリップはワークブックの右側に位置します。	vk tab strip position top	1	タブストリップはワークブックの上側に位置します。
定数	値	説明															
vk tab strip position bottom	0	タブストリップはワークブックの下側に位置します。															
vk tab strip position left	2	タブストリップはワークブックの左側に位置します。															
vk tab strip position right	3	タブストリップはワークブックの右側に位置します。															
vk tab strip position top	1	タブストリップはワークブックの上側に位置します。															
tabStripRatio	number	スプレッドシートエリアの幅において、タブストリップが占める割合を指定します(0.x)。残るスペース (1 - 0.x) には横スクロールバーが表示されます。															
tabStripVisible	boolean	タブストリップを表示します。															
tabStripWidth	number	タブストリップの位置が左側/右側の場合に、その幅を指定します。デフォルト 値は左小塎への 80 パーセント															

プロパティ useTouchLayout	タイプ boolean	説明 Spreadコンポーネントを提示するのにタッチレイアウトを使用するかどうかを指定します。
-------------------------	----------------	--

例題

"ViewProArea" 内で allowExtendPasteRange オプションを設定します:

```
var $workbookOptions : Object
$workbookOptions:= New Object
$workbookOptions.allowExtendPasteRange:=True
VP SET WORKBOOK OPTIONS("ViewProArea";$workbookOptions)
```

参照

[VP Get workbook options](#)

VP SHOW CELL

VP SHOW CELL (*rangeObj* : Object { ; *vPos* : Integer; *hPos* : Integer })

引数	タイプ	説明
<i>rangeObj</i>	Object	-> レンジオブジェクト
<i>vPos</i>	Integer	-> セルあるいは行の縦方向の表示位置
<i>hPos</i>	Integer	-> セルあるいはカラムの横方向の表示位置

説明

VP SHOW CELL コマンドは、*rangeObj* で指定したレンジの表示位置を変更します。

rangeObj 引数には、表示位置を指定するセルのレンジオブジェクトを渡します。*rangeObj* で指定したレンジの表示位置は、*vPos* および *hPos* 引数に従って縦・横に移動します。*rangeObj* の縦方向の表示位置は *vPos* 引数で、横方向の表示位置は *hPos* 引数で指定します。

次のセレクターが利用可能です:

セレクター	説明	vPos で利用可	hPos で利用可
<code>vk position bottom</code>	セルあるいは行の下辺に対する垂直揃え。	<input type="radio"/>	
<code>vk position center</code>	中央揃え。セル・行・カラムの境界に対して位置を揃えます: ● 縦方向の表示位置 - セルあるいは行 ● 横方向の表示位置 - セルあるいはカラム	<input type="radio"/>	<input type="radio"/>
<code>vk position left</code>	セルあるいはカラムの左辺に対する水平揃え。		<input type="radio"/>
<code>vk position nearest</code>	一番近い基準に対する位置揃え (上、下、左、右、中央)。セル・行・カラムの境界に対して位置を揃えます: ● 縦方向の表示位置 (上、中央、下) - セルあるいは行 ● 横方向の表示位置 (左、中央、右) - セルあるいはカラム	<input type="radio"/>	<input type="radio"/>
<code>vk position right</code>	セルあるいはカラムの右辺に対する水平揃え。		<input type="radio"/>
<code>vk position top</code>	セルあるいは行の上辺に対する垂直揃え。	<input type="radio"/>	

このコマンドは、表示位置の変更が可能な場合にのみ動作します。たとえば、`rangeObj` が現在のシートの A1 セル (先頭カラムと先頭行) の場合、すでに縦および横方向の限界に接している (つまり、上にも左にもこれ以上スクロールできない) ため、表示位置を変更しても何も変わりません。`rangeObj` が C3 セルの場合、表示位置を中央または右下に変えて同じことが言えます。表示は変更されません。

例題

AY カラムの 51 行目のセルを 4D View Pro エリアの中央に表示します:

```
$displayCell:=VP Cell("myVPArea";50;50)
// セルが表示されるよう、表示位置を調整します
VP SHOW CELL($displayCell;vk position center;vk position center)
```

結果:

	AV	AW	AX	AY	AZ	BA	BB	BC
42								
43								
44								
45								
46								
47								
48								
49								
50								
51				Hello World				
52								
53								
54								
55								
56								
57								
58								
59								
60								
61								

先ほどのコードの縦および横方向のセレクターを変更して、AY51 セルを 4D View Pro エリアの右上に表示します：

```
$displayCell:=VP Cell("myVPArea";50;50)
// セルが表示されるよう、表示位置を調整します
VP SHOW CELL($displayCell;vk position top;vk position right)
```

結果：

	AS	AT	AU	AV	AW	AX	AY	AZ
51							Hello World	
52								
53								
54								
55								
56								
57								
58								
59								
60								
61								
62								
63								
64								
65								
66								
67								
68								
69								
70								

参照

[VP ADD CELL](#)
[VP Get active cell](#)
[VP Get selection](#)

VP RESET SELECTION
VP SET ACTIVE CELL
VP SET SELECTION

VP SUSPEND COMPUTING

VP SUSPEND COMPUTING (*vpAreaName* : Text)

引数	タイプ	説明
<i>vpAreaName</i>	Text	-> 4D View Pro フォームオブジェクト名

説明

VP SUSPEND COMPUTING コマンドは、*vpAreaName* 引数で指定したエリア内の計算をすべて停止します。このコマンドは、4D View Pro エリア内の計算を停止したい場合、たとえばフォーミュラを手動で編集している際に、最終的な編集が完了するまでエラーが発生しないようにするために便利です。

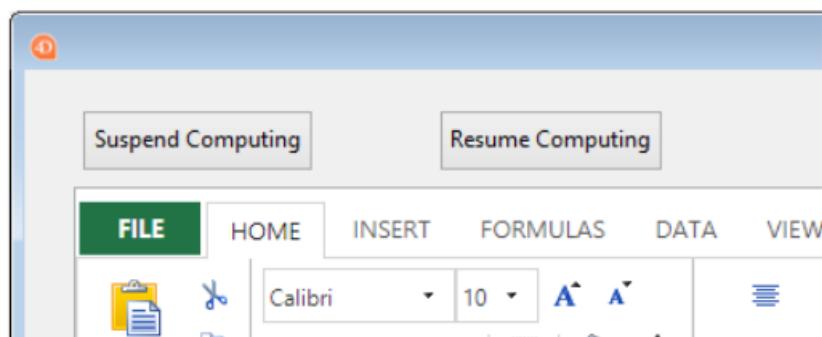
このコマンドは 4D View Pro 内の計算機能を停止します。すでに計算済みのフォーミュラはそのままですが、**VP SUSPEND COMPUTING** の実行後に追加されたフォーミュラは計算されません。

vpAreaName には、4D View Pro エリアの名前を渡します。存在しない名前を渡した場合、エラーが返されます。

4D View Pro 計算機能は停止/再開アクションを内部的にカウントしています。そのため、**VP SUSPEND COMPUTING** コマンドの実行数は、**VP RESUME COMPUTING** コマンドの実行数と釣り合っていなければなりません。計算が停止されていた間に編集された部分に影響を受けるフォーミュラは、**VP RESUME COMPUTING** コマンドが実行された時に再計算されます。

例題

ユーザーが計算を停止/再開できるように、フォーム上に 2つボタンを追加します:



計算停止ボタンのコード:

```
// ユーザーが情報を入力する間、計算を停止します
If(FORM Event.code=On Clicked)

    VP SUSPEND COMPUTING("ViewProArea")

End if
```

```
// 計算再開ボタンのコード:
If(FORM Event.code=On Clicked)

    VP RESUME COMPUTING("ViewProArea")

End if
```

参照

VP RECOMUTE FORMULAS

VP RESUME COMPUTING

Javascript による高度なプログラミング

4D View Pro Area エリアは、埋め込みWebレンダリングエンジンを使用する Webエリアフォームオブジェクト です。つまり、他のWebエリアと同様に、WA Evaluate Javascript 4Dコマンドを呼び出すことで、Javascript のコードを実行させることができます。

4D View Proは SpreadJS スプレッドシートソリューションに基づいて動作するため、4D View Pro エリア内で SpreadJS の Javascriptメソッドを呼び出すことも可能です。

例題: リボンを非表示にする

4D View Pro は Webエリアであるため、Webページ要素を選択し、Javascript を使ってその動作を変更することができます。以下の例では、spreadJS の リボン を非表示にしています：

```
// ボタンのオブジェクトメソッド

var $js; $answer : Text

$js:="document.getElementsByClassName('ribbon')[0].setAttribute(
$js+="window.dispatchEvent(new Event('resize'));""

$answer:=WA Evaluate JavaScript(*; "ViewProArea"; $js)
```

SpreadJS の Javascriptメソッドの呼び出し

Javascript メソッドの SpreadJSライブラリを利用し、これらを直接呼び出してスプレッドシートを制御することができます。

4D は、4D View Pro エリア内のスプレッドシート (ワークブックとも呼ばれます) を指すビルトインの Utils.spread 式を提供しており、これを使うことでより簡単に SpreadJS の Workbookメソッド を呼び出すことができます。

例題

次のコードは、スプレッドシートでの最後のアクションを元に戻します：

WA Evaluate JavaScript(*; "ViewProArea"; "Utils.spread.undoManager")

4D View Pro Tips のリポジトリ

[4D-View-Pro-Tips](#) は GitHub のリポジトリで、フローティングピクチャーの管理、列や行のソート、カスタムカルチャの作成など、便利な機能を満載したプロジェクトが含まれています。このリポジトリを自由にクローンして、プロジェクトで実験してください。