



HOME

User Manual

DELMIA Process Engineer®

## Scripting



# Foreword

This manual provides an introduction to the DELMIA Process Engineer basic scripting operations and functions.

While developing these functions we have made every effort to create a clearly organized, easy-to-understand program structure.

A user-friendly interface as well as a clear menu guide will enable you to quickly learn how to operate the program and to get familiar with its functions so that you can carry out your planning tasks in a quick and reliable way.

Nevertheless, there will certainly be some things that we could do even better. If you have any suggestions for improving our software, please be sure to let us know.

We look forward to receiving your constructive feedback. It helps us to make it even easier for you to work with the Process Engineer functions.

The same holds true for the manual that you are now reading. If, at any point when using these instructions, you feel you are not being provided with the clear, unambiguous, and proper guidance necessary to work with this application, please be sure to let us know. We look forward to receiving your comments and tips.

Please feel free to call, send us an E-mail, or contact our user hotline.

## **Please Send your Suggestions to:**

**DELMIA GmbH**

Raiffeisenplatz 4

D-70736 Fellbach

**Phone:** +49/711/27 300-0

**Fax:** +49/711/27 300-599

**E-mail:** [delmia.de.info@3ds.com](mailto:delmia.de.info@3ds.com)

## **User Hotline:**

If you have problems when using DELMIA products, please contact our user hotline at:

**Phone:** +49/711/27 300-400

**Fax:** +49/711/27 300-599

**E-mail:** [delmia.de.support@3ds.com](mailto:delmia.de.support@3ds.com)

## **No Liability or Guarantee**

Our programs and manuals have been compiled with great care and to the best of our knowledge. They have also been tested in a production setting. However, we assume no liability and provide no guarantee that the software and related descriptions are free of error or are suitable for special purposes.

DELMIA assumes no liability for any damage that may arise from the use of this software. By using this software, the user acknowledges this exclusion from liability and shall hold DELMIA exempt from all claims.

**Copyright**

The information in our documents may be copied and distributed for internal purposes provided it is done free of charge and the contents are not altered or distorted.

Any other form of usage, especially the sale on CD-ROM or in any other publication in whole or in part is only permitted after prior written consent by DELMIA.

Some parts of this software are owned by Unigraphics Solutions Inc. and are copyrighted © 2010. All rights reserved.

Some parts of this software are owned by combit® GmbH and are copyrighted. Report-/Print module List and Label® Version 8.0: Copyright combit® GmbH 1991-2010.

**Modifications**

Moreover, DELMIA retains the right to make modifications and improvements to the product described in this manual at any time without prior notification.

DELMIA and the 3DS logo are registered trademarks of Dassault Systèmes or its subsidiaries, in the United States or other countries.

© 2001-2010 Dassault Systèmes - All rights reserved

Thank you for your interest in our products

**DELMIA GmbH**

Raiffeisenplatz 4

D-70736 Fellbach, Germany

**Phone:** +49 (-400)711/27 300-0

**Fax:** 49/711/27 300-599

# Table of Contents

<b>1. Introduction</b>	<b>1</b>
<b>1.1 How to Use this Manual</b>	<b>1</b>
<b>1.2 Documentation Conventions and Symbols</b>	<b>1</b>
<b>1.3 New Functions in Scripting</b>	<b>2</b>
1.3.1 New Functions in PE 5.19	2
1.3.2 New Functions in PE 5.17	3
1.3.3 New Functions in PE 5.16 SP4	4
1.3.4 New Functions in PE 5.16	5
1.3.5 New Functions in PE 5.15	5
1.3.6 New Functions in PE 5.14	7
1.3.7 New Functions in PE 5.13	8
1.3.8 New Functions in PE 5.12	9
1.3.9 New Functions in PE 5.11	10
1.3.10 New Functions in PE 5.10	10
1.3.11 Changes in PE 5.9	12
<b>2. Overview</b>	<b>13</b>
<b>3. Scripting Process</b>	<b>14</b>
3.1.1 Scripting in the Process Engineer	15
<b>3.2 Managing Scripts</b>	<b>17</b>
3.2.1 Scripts, VBA Macros, and Script Actions	17
3.2.2 Location of Scripts, VBA Macros, and Script Actions in PPR-Navigator	17
3.2.3 Creating Scripts	18
3.2.4 Rights in Scripting	19
<b>3.3 Executing Scripts</b>	<b>21</b>
3.3.1 Executing Script Function	21
3.3.2 Script in the Interactive Mode	22
3.3.3 Starting Scripts Automatically by using Script Actions	22
3.3.4 Creating Real Script Actions	25
3.3.5 Creating Script Commands	26
3.3.6 Automatic Execution of Scripts	33

3.3.7 Running Scripts in Batch Mode	33
<b>3.4 Appearance of Scripts in the Process Engineer</b>	<b>36</b>
3.4.1 Error Handling	37
3.4.2 Recursions	38
3.4.3 Using Scripts in other Scripts	40
3.4.4 Logging of Script Actions	42
<b>3.5 Using VBA Host for Programming</b>	<b>42</b>
3.5.1 Introduction	42
3.5.2 Creating VBA Scripts	43
3.5.3 Changing Scripts into VBA Macros	46
3.5.4 Error Handling with VBA Macros	48
<b>4. List of Script Actions</b>	<b>50</b>
<b>4.1 Script Actions</b>	<b>52</b>
4.1.1 ScriptActionNewChild	52
4.1.2 ScriptActionAddChild	52
4.1.3 ScriptActionSetAttribute	53
4.1.4 ScriptActionNew	54
4.1.5 ScriptActionCopy	54
4.1.6 ScriptActionLink	55
4.1.7 ScriptActionMove	56
4.1.8 ScriptActionDelete	57
4.1.9 ScriptActionRemoveChild	58
4.1.10 ScriptActionInitProperties	59
4.1.11 ScriptActionChangeProperties	60
4.1.12 ScriptActionNewVersion	61
4.1.13 ScriptActionSetPlanningState	62
4.1.14 ScriptActionOpenProject	63
4.1.15 ScriptActionCloseProject	63
4.1.16 ScriptAction SelectProject	63
4.1.17 ScriptActionStartBalancing	64
4.1.18 ScriptActionSaveBalancing	64
4.1.19 ScriptActionDropProcessToBalancing	65
4.1.20 ScriptActionPrint	65
4.1.21 ScriptActionPrintlist	66
4.1.22 ScriptActionOpenGraphic	66
4.1.23 Script Action RemoveChildEx	67

4.1.24	Script Action Select Tree Item	68
4.1.25	Script Action Expand Tree Item	68
<b>5.</b>	<b>Class Documentation XScriptingHost</b>	<b>69</b>
<b>5.1</b>	<b>Class ScriptItemData</b>	<b>70</b>
5.1.1	List of ScriptItemData Methods	71
<b>5.2</b>	<b>Class ScriptItemRights</b>	<b>101</b>
5.2.1	List of all XScriptItemRights Methods	102
<b>5.3</b>	<b>Class ScriptItemGrid</b>	<b>119</b>
5.3.1	List of XScriptItemGrid Methods	119
<b>5.4</b>	<b>Class ScriptItemGraphic</b>	<b>120</b>
5.4.1	List of XScriptItemGraphic Methods	120
<b>5.5</b>	<b>Class ScriptItemDialog</b>	<b>129</b>
5.5.1	List of XScriptItemDialog Methods	130
<b>5.6</b>	<b>Class ScriptItemList</b>	<b>148</b>
5.6.1	List of XscriptItemList Methods	148
<b>5.7</b>	<b>Class ScriptItemQuery</b>	<b>150</b>
5.7.1	List of ScriptItemQuery Methods	151
<b>5.8</b>	<b>Class ScriptItemConfig</b>	<b>165</b>
5.8.1	List of XScriptItemConfig Methods	165
<b>5.9</b>	<b>Class ScriptItemConvert</b>	<b>170</b>
5.9.1	List of XScriptItemConvert Methods	171
<b>5.10</b>	<b>Class ScriptItemVersion</b>	<b>172</b>
5.10.1	List of XScriptItemVersion Methods	173
<b>5.11</b>	<b>Class ScriptItemUnit</b>	<b>185</b>
5.11.1	List of XScriptItemUnit Methods	186
<b>5.12</b>	<b>Class ScriptItemLock</b>	<b>191</b>
5.12.1	List of XScriptItemLock Methods	192
	<b>List of Figures</b>	<b>197</b>
	<b>List of Tables</b>	<b>199</b>
	<b>Index</b>	<b>200</b>

# 1. Introduction

This manual explains how to use the Process Engineer scripting functions and menu guidance for your planning purposes.

## 1.1 How to Use this Manual

This manual enables you to get familiar with the operation and functions of the basic settings. This manual briefly describes:

- Xscripting host classes
- How you can use and utilize the scripting for the different application programs in the Process Engineer



### Note

*When handling the scripting functions, please remember that there is a general introduction to the Process Engineer in the Basic Manual.*



Click [General Introduction](#) to access the manual

## 1.2 Documentation Conventions and Symbols

The symbols used in this manual are intended to provide you with keys to the contents in an immediately understandable manner.



This symbol is used to introduce key concepts that are covered in the sections immediately following this symbol. As a result, this symbol most frequently appears at the beginning of chapters or sections.



### Note

*This symbol is used to mark notes, which provide you with additional information you need to have for further work. You will either find the Note sign at the beginning of a chapter or in a particular text passage in the chapter. Texts bearing this sign are additionally marked with **Note**. The text is always in italics.*




### Caution

*This symbol indicates that the text that follows describes particular circumstances that you must avoid to avoid potential errors with the operation of the program or harm to data. You will either find the Caution sign at the beginning of a chapter or near a particular text passage in the chapter. Texts that are introduced by this sign are additionally marked with **Caution**. The text is always in italics.*

**Example**

This symbol marks examples which serve to illustrate a certain situation.

- 1) This symbol marks the individual operational steps involved in a particular operating instruction. Operating instructions describe operational steps, for example, how to open a menu or execute a function.
- This symbol marks listed subjects. The symbol for listed subjects can be either used to structure a continuous text or to list main subject keywords.
- This symbol marks list inside a bulleted or numbered list.
-  This symbol marks cross reference information that is available in another manual.

## 1.3 New Functions in Scripting

### 1.3.1 New Functions in PE 5.19

- [DeleteVersionForcedEx\(\)](#)

A new API method `DeleteVersionForcedEx()` for the interface `IEPVersion` allows users with a particular function right to delete CCZ Members and its CCZ children versions.

- [Select Tree Item](#)

After an object is selected (or selection changed) by user in **PPR Navigator Tree** a script action (`sa_selecttreeitem`) would be executed in order to better control the customized workflow.

- [Expandt Tree Item](#)

After an object is expanded in **PPR Navigator Tree** a script action (`sa_expandtreeitem`) would be executed in order to better control the customized workflow.

- [ScriptActionRemoveChildEx](#)

Script `sa_removechild` is now extended to `sa_removechildex` for supporting “*simplerelation*” as well.

#### Differences

From R19sp5 the following functions will respect access-rights if client authentication is enabled:

- [Query.GetNextResult](#)
- [Query.GetFirstResult](#)
- [Query.GetOnlyFirstResult](#)
- [Query.GetResultCount](#)



- [Data.GetFirstChild](#)
- [Data.GetNextChild](#)
- [Data.GetChildrenCount](#)

The returned ID's and count will correspond to objects having at least read rights.

Registry entry details:-

```
key = HKEY_LOCAL_MACHINE\SOFTWARE\DELMIA\Ergoplan System\Security
name = ClientAuthenticationEnabled
value = 0
"0" - No right checking takes place when value = 0
"1" - Rights checking takes place when value = 1
```



### Caution

*Implementation added for the above mentioned functions to check for user rights on each resultant object. This involves server calls to check for user rights on each object, this will impact the performance.*

*Where as if the value is "0" (ClientAuthenticationEnabled is set to "0") respects the old behavior and there is no impact in the performance.*

## 1.3.2 New Functions in PE 5.17

### 1.3.2.1 Extended Categories

- Extended script command properties (*Please refer to the [Creating Script Commands](#)*)
  - As of version PE 5.17 script assignments can be defined for individual pages, groups, and attributes.
- Added the function Data [CreateRelationshipEx](#).
- Added the function Version [UpdateRelations](#).
- Connected with the scriptactions [ScriptActionChangeProperties](#) and [ScriptActionInitProperties](#)
  - [PropModifyVisibility](#) (Hide/Show customization layout objects)
  - [PropModifyAccess](#) (Disable/Enable attribute access)
  - [PropModifyBGColor](#) (Change background color)
  - [PropModifyFGColor](#) (Change foreground color)
  - [PropModifyFontSize](#) (Change font size)
  - [PropModifyFontStyle](#) (Change font style)
  - [PropModifyFontType](#) (Change font type)
  - [PropModifyRep](#) (Set the representation of a attribute)
  - [Redraw](#) (Redraw all pages)
  - [RedrawPage](#) (Redraw page)
  - [RedrawGroup](#) (Redraw group)

- [RedrawAttribute](#) (Redraw attribute)
- Scripting methods to access the OpenProject dialog: Connected with the new scriptaction [ScriptAction SelectProject](#)
- [GetValueCB](#) (Get combobox value)
- [GetValueEdit](#) (Get edit value)
- [GetValueCKB](#) (Get checkbox value)
- [SetValueCB](#) (Set combobox value)
- [SetValueEdit](#) (Set edit value)
- [SetValueCKB](#) (Set checkbox value)

### 1.3.3 New Functions in PE 5.16 SP4

#### 1.3.3.1 New Functions

- Introduction of the additional script action [ScriptActionPrintlist](#)
- Added the functions Data [ExecuteServerMethodEx](#)
- Added the functions Graphic [ShowGraphic](#)
- Added the functions Rights [GetRightSubjects](#)
- Added the functions Version [Promote](#)
- Added the functions Version [CanPlanningStateBeModified](#)

#### 1.3.3.2 Differences

- Extended script command properties (*Please refer to the [Creating Script Commands](#)*)
- Now it is possible to specify whether an associated script has to be executed for each single object of a list multi selection or once for all
- An entry in the context menu is created
- An execution button is created in the associated objects
- [ScriptActionChangeProperties](#) (sa\_changeproperties): The "Change Properties" script action which was confined to combo box and list box controls, is now extended to the following controls:
  1. Edit
  2. Multi Line Edit
  3. Check Box
  4. Radio Buttons
  5. Date Control

## 1.3.4 New Functions in PE 5.16

### 1.3.4.1 New Functions

The following functions has been added:

- [CreatePreviewGraphic](#) (Graphic files are created and saved in a special directory)
- [AttributeExists](#) (Checks existence of an attribute on a particular object)
- [CreateRelationshipEx](#) (Creates a Relationship – between objects (exposed or unexposed) – with or without owner)
- [ChangeRelationship](#) (Changes participating objects of a relationship)
- [UnlockAllObject](#) (To release all locks at once)
- Regarding the shop floor integration (SFI) some new functionality are introduced in PE5.16. Concerning the handling of ReleaseTables there are new features:
  - [CreateReleaseTable](#) (For first creation)
  - [GetReleaseTable](#) (For allocation of an existing release table)
  - [PDXMLFileCreated](#) (For setting the marker that a *pdxml* – file has been created for an existing releasetableentry)

The existing methods:

- [Create](#)
- [CreateDeep](#)
- [CheckOut](#)
- [CheckOutDeep](#) consider the new introduced SFI consistency checks. They can be overridden by the new methods.
- [CreateSFI](#)
- [CreateDeepSFI](#)
- [CheckOutSFI](#)
- [CheckOutDeepSFI](#)
- Introduction of the additional script action [ScriptActionDelete](#) (After) PE5.15 and later.
- Recursive fetching of children available (application server side change) R15 and later.

## 1.3.5 New Functions in PE 5.15

### 1.3.5.1 New Functions

The following functions has been added:

- [IsDerivedFromType](#) (Replaces Data.IsDerivedFromClass)

- **AttributeExists** (Check existence of an attribute on a particular object), R15SP3 and later.
- **CompLock.UnlockAllObjects** (To release all locks at once), R15SP3 and later.
- Introduction of the additional script action **ScriptActionDelete** (After) PE5.15 and later.
- Recursive fetching of children available (application server side change), R15SP3 and later.

### 1.3.5.2 Differences

From a functional point of view there are almost no differences compared to release R14. Some major changes on the application server side make some script and command line adaptations mandatory, due to introduction of Single Sign-On (SSO) and encryption of passwords.

- The command line for batch execution has to be changed. *Please refer to the [Running Scripts in Batch Mode](#)*
- On creation of a new user the password string has to be encrypted before setting the password attribute. *Please refer to the [Create](#).*
- Regarding queries, due to a major change of the query interface on the application server side, passing of physical names (For example, "XDOErgoCompProcessdefault", "m\_pErgoProject", etc.) as parameters in functions of the Query object is not possible anymore. Instead you have to pass configured names (For example, "ergocompprocessdefault", "ergoproject", etc.).
- The class **XDOSubCompltem** is now derived from **XDORelationship** for ease of data model design. This requires some changes, especially in for the Query script item. The Data script item is not affected that much, because some of the calls will be redirected in the server automatically to account for data upward compatibility.
- **Data.GetChildren**(parent\_id, "subcompitem") is not supported anymore. Use **Data.GetChildren**(parent\_id, "nodes") instead.
- A query on "XDOSubCompltem" is not possible anymore. Instead, the correct table name is "relationship\_nodes". Moreover, replace attribute names "m\_pDODefaultImpl" or "m\_pErgoCompBaseParent" by "relationobjectsource" and "m\_pErgoCompBase" by "relationobjecttarget".
- **Data.GetAttributeById**(sci\_id, "ergocompbase") still works, it is mapped to "relationobject2" on the server side. For a better performance, it is recommended to call **Data.GetAttributeById**(sci\_id, "relationobject2") directly. "relationobject2" is the equivalent of "relationobjecttarget" in the query.
- **Data.GetAttributeById**(sci\_id, "ergocompbase\_parent") still works, it is mapped to "relationobject1" on the server side. For a better performance, it is recommended to call **Data.GetAttributeById**(sci\_id, "relationobject1") directly. "relationobject1" is the equivalent of "relationobjectsource" in the query.



#### Note

*The sample scripts of this documentary still use "ergocompbase" and "ergocompbase\_parent".*

- In case a script action is disabled and logging is enabled, no log entry is written anymore for performance reasons.
- When using the Query script item, make sure to provide the name of the query table once for each query (the main query and each of the subqueries), even if the name does not change for the next function call.

### 1.3.5.3 Known Restrictions

- The "Change Properties" script action is currently confined to combo box and list box controls

## 1.3.6 New Functions in PE 5.14

### 1.3.6.1 New Functions

File based batch execution of scripts (in addition to script object based batch execution).

The following functions has been added:

- Data.DisableChildrenListFilter (To get unfiltered children lists)
- Data.SetAttributeRange (To get a list of objects instantiated from a distinct plantype within a predefined attribute value range)
- Data.ReadSessionData (To get read access to session data) - similar to **Tools < Settings < Maintenance Tool ...**)
- Data.WriteSessionData (To get write access to session data)
- Introduction of one additional script action
  - Open Graphic (Before)

### 1.3.6.2 Differences

Due to a data model change, objects of type:

- "subcompitem" (BOM entries) and "subcompviewitem" (including BOM views) do not implement a pointer "m\_pBom" (configured name "bom") anymore.

Scripts which make explicit use of the attribute "bom" when calling Data.GetAttributebyId or use the attribute "m\_pBom" in queries have to be revised.

The function call:

- `bom_id = Data.GetAttributebyId(sci_id, "bom")` is no more valid
- On the other hand, the implementation
- `parent_ecb_id = Data.GetAttributebyId(sci_id, "ergocompbase_parent")` does not only remain valid, instead of a special server implementation it now represents an existing physical link in the data model. With other words, for PE5.14 and later, a BOM entry references its parent base component directly.

In queries, please replace functions similar to:

```
bom_id = Data.GetAttributebyId(parent_base_id, "bom")
call Query.SetQuery("subcompitem", "m_pBom", "=", bom_id)
```

- by

```
call Query.SetQuery("subcompitem", "m_pErgoCompBaseParent", "=",
parent_base_id).
```

The BOM entries model might be subjected to further revisions in later versions. It is likely that BOM entries are replaced by special relationships in the future.

### 1.3.6.3 Known Restrictions

The "Change Properties" script action is currently confined to combo box and list box controls.

## 1.3.7 New Functions in PE 5.13

### 1.3.7.1 New Functions

- Introduction of two additional script actions related to the **Properties** dialog
  - Init Properties
  - Change Properties

Using these script actions and the related new functions

- XscriptItemDialog. PropSetValues
- XscriptItemDialog. PropGetValues
- XscriptItemDialog. PropGetCurrentValue

It is possible to implement properties dialog inter-control workflow. This means that the content of a control may depend on the value selection of another control and can be changed on-the-fly, i.e. when the **Properties** dialog remains open.

#### Creation of Children Objects on Root Objects

With PE 5.13 functions are provided to create "create children" on root objects.

- ProjectRoot (creating projects)
- ArchivRoot (creating objects in the global library)
- Data.CreateRootOrLibraryComponent
- RightsRoot" (creating users and groups)
- XscriptItemRights. Create
- XscriptItemRights. GetFunctionRights

#### Script-Defined Finder Initialization

Executing a script, it is now possible to start and initialize the project finder window with a predefined context window.

- XscriptItemList. OpenFinder

### 1.3.7.2 Differences

At type *XDORelationship*, the attribute

- "**m\_pOwnerErgoComponent**" has been replaced by "**m\_pOwner**"
- Attribute "**owner**" was renamed to "**relationowner**"

If you are using "m\_pOwnerErgoComponent" and "ownerergocomponent" explicitly in a script, you have to change this to "m\_pOwner" and "relationowner" respectively.

**Reason:** There was a naming conflict with the "owner" attribute in type "dodefaultimpl", "relationowner" returns the **owner component** of a relation (an object ID), while "owner" at type "dodefaultimpl" returns the **name** of the user who owns an object.

#### Known Restrictions

- The "Change Properties" script action is currently confined to combo box and list box controls.
- Queries do not take into account the user-defined attributes.

## 1.3.8 New Functions in PE 5.12

### Run Scripts using the Application Server Locale

In multi-client environments with different locale settings on clients and/or server, it may be required to run scripts using the locale of the server, because otherwise setting distinct object attributes does not work. Using the application server locale can be enabled via the settings. The client locale is temporarily set to the server locale, during runtime of a script. It is recommended to set this option only if required.

### Multi-Selection Script Execution

On selecting multiple nodes in a browser list view, the selected script is executed on each of the nodes subsequently.

### 1.3.8.1 Changes in PE 5.12



#### Note

*The following changes in the server infrastructure area potentially require adaptation of existing scripts.*

### Rights Database Integrated into the Object Database

Permissions are now part of the object database. The separate rights database has been removed. As a consequence, many of the functions and methods of the *IScriptItemRights* object had to be changed. Groups, users, object, and plantype rights are now handled like any other data object. This particularly means that rights object do behave in the same way as other object does. A rights object has an object ID and once you have this object ID you can ask for the attributes of a rights object. The former ID of users and groups are stored in the attribute "rightsubjectid" (for compatibility reasons only).

### Avoid Usage of Real Attribute Names in Get/SetAttribute(s)byId Calls

Although the attribute *m\_pBom* does exist, it is not possible to use it within *Get/SetAttribute(s)byId* calls anymore. Due to performance reasons, the implementation of the server *Get/SetAttribute* functions has been completely switched to lower case attribute names. When you pass the string "*m\_pBom*"

this get converted to m\_pbom and thus the query on the database - which is case sensitive - does not recognize m\_pbom as valid. However, using the string identifier "bom" works.

### **Relationship Names have to be Specified with Direction**

You have to take into account the **direction** of relationships when using relationship names in scripts.

From the viewpoint of the the process, one of the relationship names to get related products is "proc\_creates\_prod". But, seen from the product, use "proc\_creates\_prod\_reverse" to retrieve the processes, respectively.

## **1.3.9 New Functions in PE 5.11**

- New Script Functions: Three new script actions have been added which are used by the balancing module:
  - Start Balancing
  - Save Balancing
  - Drop Process(es) To Balancing
- XscriptItemData.CopyAutoRelationUsageData

## **1.3.10 New Functions in PE 5.10**

- Project independent scripts and Script Actions
- Adding scripts to other scripts
- Scripts in own transaction
- The Visual Basic for Applications (VBA) Integration
- New Classes: There are the following new classes:
  - XScriptItemConfig (the access to the configuration database)
  - XScriptItemUnit (the access to the units)
  - XScriptItemLock (Lock/Unlock objects by means of a script)
  - XScriptItemConvert (Conversion functions)
- Changed (extendend) classes are:
  - XScriptItemRights (Reading and specifying object, type and function rights)
  - XScriptItemData (Conversion of object ID into object GUID)
  - XScriptItemQuery (Access to filtered database requests, execution of OQL database queries)

### **1.3.10.1 Changes in PE 5.10**

- GetFirstChild/GetNextChild

As changes have to be made to the Scripting Host, you have to **rework existing scripts**.



The FirstChild/GetNextChild call has been generated by a Subcompitem ID until now. Subcompitem refers to ergocompbase type-objects with their corresponding child lists. The Scripting Host has redirected the Subcompitem function call to the ergocompbase type in an implicit way.

After R10 upgrading, there are usually problems with existing scripts in **convoluted** or **recursive** BOM-structures because the returned child IDs are *subcompitem* IDs.

```
REM parent_sci_id: ID of a subcompitem
REM parent_id: ID of the referenced ergocompbase

REM Invalid code for R10 and later
child_id = Data.GetFirstChild(parent_sci_id, childlistname)
Do while child_id <> ""
    child_id = Data.GetNextChild(parent_sci_id, childlistname)
Loop

REM Valid code for PE 5.10 and later
parent_id = Data.GetAttributebyId(parent_sci_id, "ergocompbase")
child_id = Data.GetFirstChild(parent_id, childlistname)
Do while child_id <> ""
    child_id = Data.GetNextChild(parent_id, childlistname)
Loop
```

## Example

```
REM *****
REM ***Recursive Reading of children***
REM *****
sub main(entryid)
GetSCIChildren(entryid)
end sub
sub GetSCIChildren(id)
    if id <> "" then
        parent_id = GetBase(id)
        child_id = Data.GetFirstChild(parent_id, "nodes")

        Do while child_id <> ""
REM Recursion
            GetSCIChildren(child_id)
            child_base_id = GetBase(child_id)
            name = Data.GetAttributebyId(child_base_id, "name")
            MsgBox(name)
            child_id = Data.GetNextChild(parent_id, "nodes")
        Loop
    end if
end sub

function GetBase(id)
    GetBase = Data.GetAttributebyId(id, "ergocompbase")
end function
```

- The “Script Actions” folder in the project plantype set (former project library)

Since the introduction of the plantype set pool in the system library, scripts, and script actions are no longer dependent on a project.

The script action functions have not changed due to the movement of script actions from the project library to the project plantype set.

In general, script actions no longer belong to a project, but are now dependent on the plantype set. Each script action can be linked to any plantype of the plantype set which the script action belongs to – one could say that plantypes

and script actions are brothers and sisters and the children of a plantype set (parents). As a consequence, project-independent script actions belong to one of the system library plantype sets.

### 1.3.11 Changes in PE 5.9

Changes affect: [SetAttributebyId](#) / [SetAttributesbyId](#)  
[/SetLinkedObjectAttributebyId](#) / [SetLinkedObjectAttributesbyId](#)

When being set, an attribute of a **subcompitem** component no longer refers to the **ergocompbase** of this component. This is the reason why you have to inquire about the attribute of the corresponding ergocompbase component before you start a **set**-call of a subcompitem component.

#### Example

1) The **call** function in the PE 5.7 and 5.8 requires a basic object (base\_id).

```
call Data.SetAttributebyId(sci_id, "name", "New Name")
```

2) From version PE 5.9, the **call** function needs to be called as follows:

```
base_id = Data.GetAttributebyId(sci_id, "ergocompbase")  
call Data.SetAttributebyId(base_id, "name", "New Name")
```

## 2. Overview

Modern operating systems offer means and ways to automate routine tasks by means of scripts. Batch programming on the basis of command lines which is restricted to a small number of options has long been the only possibility to do this. The introduction of a powerful script interpreter, the Windows Scripting Host (WSH) as a standard, has overcome this deficiency.

You can use the script language interpreters installed on your system to apply the broad range of possibilities offered by scripting in the Process Engineer. VBScript (Visual Basic Script) and JScript (Java Script) system libraries provided with MS Windows and/or MS Internet Explorer enable script language usage. Experience shows that most users write their scripts in Basic which is why the examples shown in the manual are written in VBScript.

Additional script languages such as Perl, Tel, Python, or REXX can be integrated in the WSH by means of the corresponding language engines. The language that you use is dependent on your knowledge and the language you prefer.

## 3. Scripting Process

### Introduction to Scripting

Scripting can also be defined as programming! The main purpose of scripting is to enable you to specifically upgrade an operating system and programs. This mainly applies to smaller tasks, but its complexity, however, is only limited by the script language being used and the options offered for interaction with programs and the operating system. Scripting, therefore, helps you control your applications in a flexible manner and automates routine tasks.

### Introduction to Windows Scripting Host

The Windows Scripting Host is responsible for the execution of the scripts you have written, i.e. it interprets the script which is written in a particular language and transmits the corresponding commands to the operating system as well as the applications and objects used. The WSH first checks the script syntax i.e. if there is spelling mistakes or logical faults. After this, it executes the code.

### Introduction to Scripts Writing

You can use any kind of editor for writing a script. Short scripts can be written using a simple text editor like *Notepad* or *Wordpad*. For longer scripts we recommend using an editor that allows so-called *Syntax highlighting*, i.e. language-specific elements are structured (colored/bold/block by block) thereby providing you with a clear overview.

### Executing a Script


Generally, it is sufficient to use a text editor for writing a script and to save this file with an ending that WSH can recognize. Default endings are \*.vbs for VBScript and \*.js for JScript. The WSH interprets a file with a \*.vbs ending as a VBScript and tries to execute it. However, things are a bit more complicated when it comes to the communication with the *Process Engineer* objects. Scripts which communicate with the *PPR Hub* can only be executed within the *Process Engineer* – the reasons for this will be dealt with later. Hence, this limitation was deliberately set.

### To Execute a Script

- 1) Open *Notepad* or *Wordpad* (text editors) and enter the following line.

```
MsgBox "Hello, this is an output box", , "My first script"
```

- 2) Save this entry as MyScript.vbs.

Afterwards, the file-icon should look like this: 

- 3) Doubleclick the file to execute the script. The result should be as follows:



**Figure 1: Script Example**

The *MsgBox* method is part of the VBScript programming language. Just like any other programming language, VBScript contains a large number of *functions and control structures* (loops, branches) for processing data which

are transmitted as *constants* and *variables*. A script first becomes powerful, however, by the communication with *objects* provided by the operating system and other applications. The *FileSystemObject* is one of these objects that allows *VBScript* to communicate with the file system, which, in particular, allows for the reading and writing of files. The following source code generates a c:\test.txt file which generates the line. This text file has been generated by a script.

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set ts = fso.CreateTextFile ("c:\test.txt", ForWriting,True)
ts.WriteLine("Diese Textdatei wurde per Skript generiert.")
ts.Close
```



For more information, *Please refer to the* [Administration Manual](#).

### Security Aspects with Script Usage

Computer viruses such as *Melissa* or *Code Red* are discussed everywhere. They have negatively affected scripts and script interpreters because these viruses are none other than scripts used for sabotage. The easiest solution is to switch off all interpreters. If this is the case with you, then you can stop reading right now because you simply *cannot* execute scripts. Most Internet/Intranet pages are not displayed at all or displayed incorrectly. There is, as is true for human viruses, no 100% protection against computer viruses – unless you seal yourself off from the rest of the world. In the computer world this is equivalent to switching off the script interpreter. If you want to protect yourself against (script) viruses without having to seal yourself off from the rest of the world, you are recommended to ensure that the scripts you are executing have either been written or checked by you personally or that they are provided by a reliable source.

## 3.1.1 Scripting in the Process Engineer

### 3.1.1.1 System Requirements

System requirements are not particularly high:

#### Windows NT 4

Service Pack 3 and higher

Internet Explorer 4.01 or higher

The Windows® Scripting Host (WSH) has to be installed and activated

#### Windows2000® und Windows XP®

No further requirements

#### DELMIA Process Engineer®

PE 5.7 and higher



### Note

*If, at a later point, the text deals with the opening or accessing of applications other than DELMIA Process Engineer, it is assumed that these tools also exist in the corresponding system. If, for example, data are transferred to MS-Excel, MS-Excel needs to be installed on the system, of course, so that the script can be executed correctly.*

### 3.1.1.2 Scripting Interface of the Process Engineer

The Process Engineer provides its own interface enabling access to the PPR-Hub from a particular subset of access methods for the purpose of scripting. The individual classes are as follows:

ScriptItemData, ScriptItemQuery, ScriptItemVersion, ScriptItemList, ScriptItemDialog, ScriptItemGraphic, ScriptItemGrid, ScriptItemRights, ScriptItemUnit, ScriptItemConvert, ScriptItemConfig, ScriptItemLock.

For a detailed description of the classes, *Please refer to the [Error! Reference source not found.](#)*



#### Note

*The subsequent versions of the Process Engineer provide further **named items**; in addition, the range of functions of the existing items is extended.*

*Objects provided by the DELMIA Process Engineer can only be accessed by scripts which are executed within the Process Engineer. Hence, you cannot use the DELMIA Process Engineer Object "data" in any \*.vbs-file you like.*

Each of these COM classes provides a dual interface derived from the IDispatch which provides the corresponding methods.

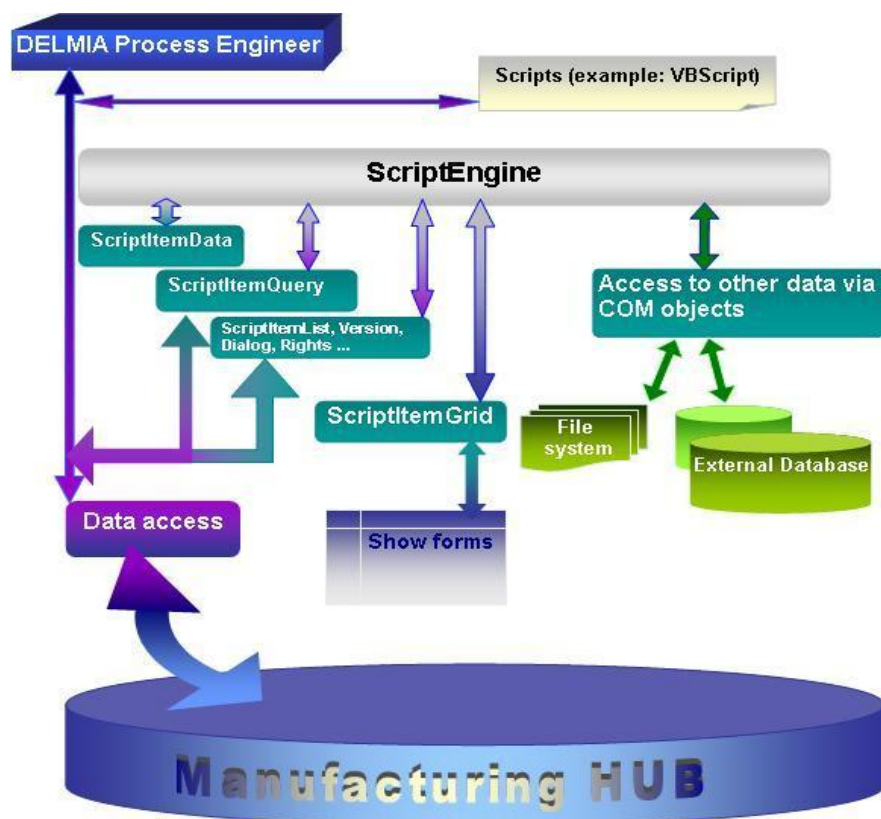


Figure 2: Interaction between the Script Engine and the DPE



#### Note

*To be able to use the attribute names, the names of the parent-child-relations and the child lists, the scriptwriter needs to know the scriptlanguage as well as the configuration tool of the Process Engineer.*

### 3.1.1.3 Scripting Requirement in the Process Engineer

The scripting requirement occurs in the process Engineer in the following cases:

- Any data changes in the DELMIA Process Engineer.
- Using any mathematical algorithms or functions.
- Exporting or importing of data.
- Checking data structures that are read in from other data systems.
- Filtering of particular data information with corresponding information on the monitor.

## 3.2 Managing Scripts

### 3.2.1 Scripts, VBA Macros, and Script Actions

You can create and use Macro codes in various ways in the DELMIA Process Engineer®. Macrocodes can be used by the following objects:

- Scripts
- VBA Macros

**script**-type objects are saved in the database whereas **VBA macros** belonging to the VBA Project object are saved within a container.

- **Script Actions** are not scripts; they are objects which, in connection with scripts, are used for two different purposes:
  - Script Commands: it is the configurable display of scripts in context menus. It links a script to a configuration type or plantype.
  - Actual (real) Script Actions: it is the action-driven, automatic execution of the scriptcode. It links a script, a configuration type or a plantype to an action.

VBA Macros cannot be assigned a real script action.

### 3.2.2 Location of Scripts, VBA Macros, and Script Actions in PPR-Navigator

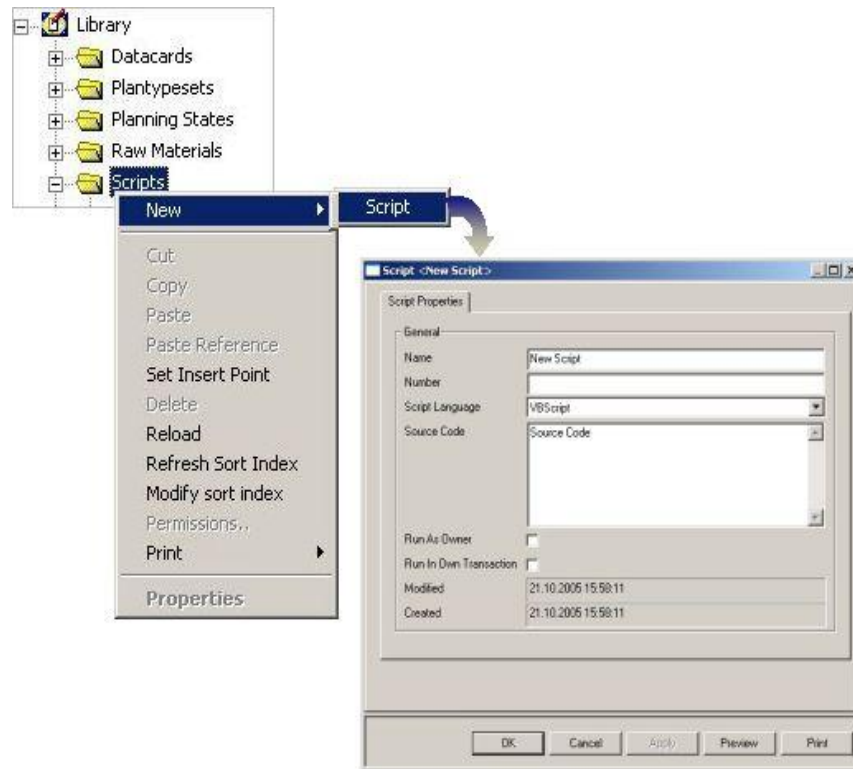
Table 1: Location of Scripts, VBA Macros, and Script Actions in PPR-Navigator

Script/VBA Macro/Script Actions	Location
Scripts	Project library System library
Script Actions	Project plantype set in each library plantype set
VBA Macros	These are part of a VBA Project –in the project library only

Script objects can be found in the **Scripts** folder of the **Project library** and in the **System library**.

### 3.2.3 Creating Scripts

- 1) You can find a “Scripts” folder in the system library. This folder does not contain any entries during the initial installation. To fill the folder, rightclick the context menu and select the **New < Script**. The script editor gets open.
- 2) In the script editor, you can specify the script name and enter the script code.
- 3) To create scripts in the project library, proceed in the same way.



**Figure 3: Creating a New Script**

You do not necessarily have to enter your script text in the DELMIA Process Engineer®, of course.

It does not matter where and by which means you edit your scripts – the important thing to do is to enter your script code in the “source text” field of your script object – either by direct editing or by using the Copy and Paste function.



#### Note

*If the file tools does not appear after a new installation, you have to make it appear with the help of configuration tools. How to show the file, is written in the [Administration Manual](#).*



Figure 4: Script Properties

### 3.2.3.1 Parameter Description

#### Name

Enter the script name here. You are able to retrieve the script later under this name.

#### Number

Enter the script number here.

#### Script Language

Here, you specify the script language for writing the script. As default language there is either: VBScript or JScript.

#### Source Code

Enter the script code here.

The script size (for ORACLE) is dependent on the database properties.

#### Runn As Owner

This function provides you with further rights for the execution of a script. *Please refer to the [Rights in Scripting](#).*

#### Run In Own Transaction

You can start a script in your own transaction, i.e. the script works in an environment independent from or other than the Process Engineer. *Please refer to the [3.2.4.2 Running Scripts in own Transaction](#).*

## 3.2.4 Rights in Scripting

The **Run As Owner** function provides the user with additional rights for the execution of a script. *Please refer to the Configuration section in the [Administration Manual](#).*

### 3.2.4.1 Executing a Script as a Script Owner

In previous versions, the script writer of a script was the script owner as well. In these previous versions, the owner had the user rights needed for the creation and execution of a script, but access rights to particular plantypes were limited. As a consequence, failures might occur with the execution of scripts.



For more information, *Please refer to the* [User Management](#) section in the [Administration Manual](#).

With the **Run As Owner** function, you can now start a script independently of the execution type as if the script owner were the present user. The following rights are to be granted:

- The right to create, write, and change scripts.
- The right to execute interactive scripts.

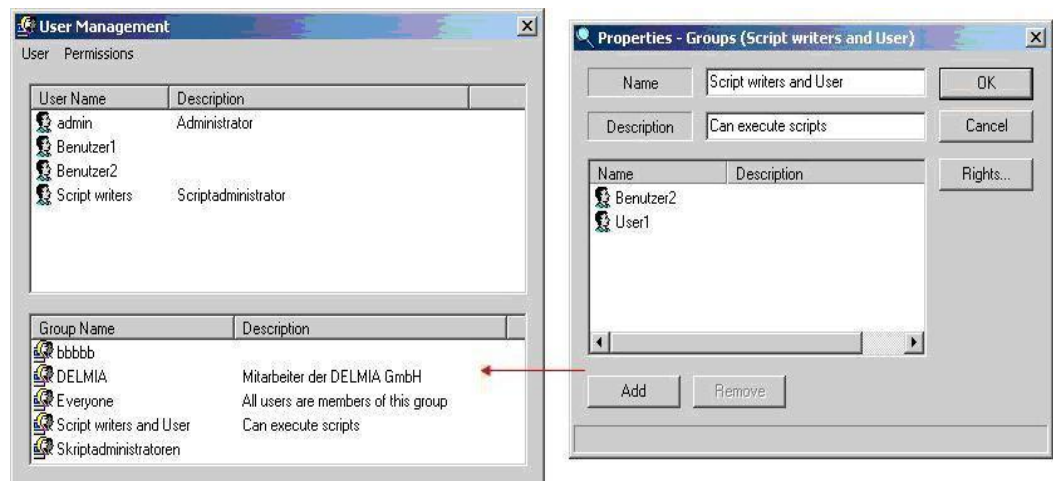
The script writer has administrator rights; these rights are transferred to the script executor by the **Run As Owner** function.

- 1) Activate the **Run As Owner** function in the script dialog. *Please refer to the* [Figure 4](#).



#### Note

*The introduction of the Run As Owner function represents an additional task for the administrator. For this reason, administrators are recommended to create a group for the users who are entitled to write and execute interactive scripts.*



**Figure 5: User Management**

Is there a problem if a normal user is granted to create own scripts, Not really. Whenever a normal user creates a script, he/she is the owner of the script and hence, even if "RunAsOwner" is enabled the script - as every script using the *RunAsOwner* feature - observes the rights of this user.

### 3.2.4.2 Running Scripts in own Transaction

You can now start a script in an own transaction.

Run In Own Transaction

This is particularly useful when executing a script which can generate critical changes. If errors occur during the execution, the changed data is not saved.

This function can be activated for each script in the **Properties** dialog of a script.

### Running a Script in an own Transaction

- Changes made by the script are not saved during the run time of the script.
- If errors occur during script execution, the script is closed without saving. Changes made up until the time when the error occurred, are not saved in the database.
- Changes are not saved until the script is finished correctly.

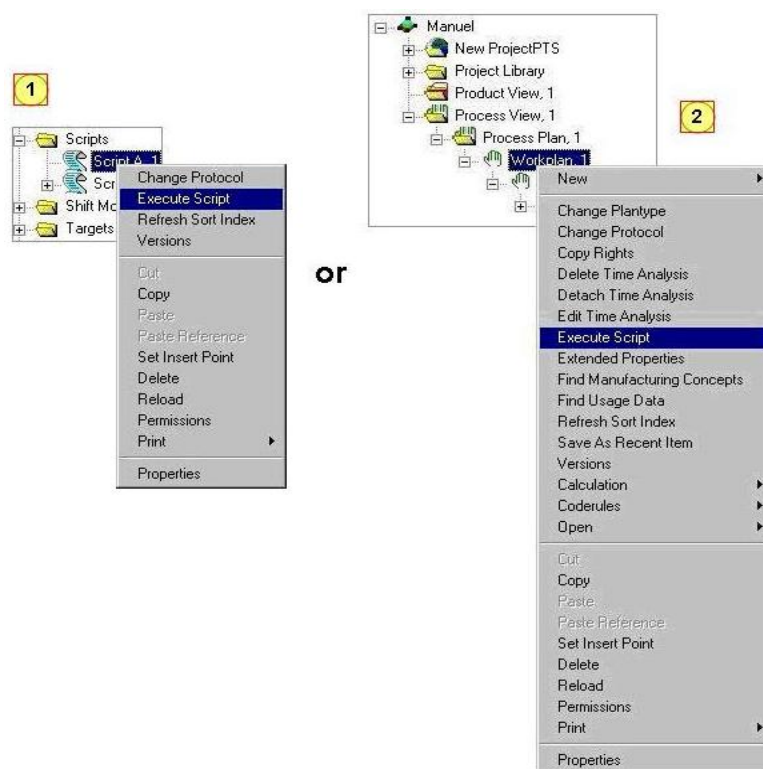
If you have activated the function and the script is active, you cannot see online changes in the PPR Navigator until you call a `Data.CommitTransaction` command in your script.

## 3.3 Executing Scripts

There are different ways to start a script.

### 3.3.1 Executing Script Function

A script is executed in the context menu of a project or a project component. You find an **Executing script** menu item for each script. If you select this menu item, you get a selection of all scripts assigned to the project and which you are entitled to execute. Select the required script from this menu.



**Figure 6: Option for starting a Script**

A window with all available scripts shown in a list gets open. If you confirm the selection by **OK**, the script is started.

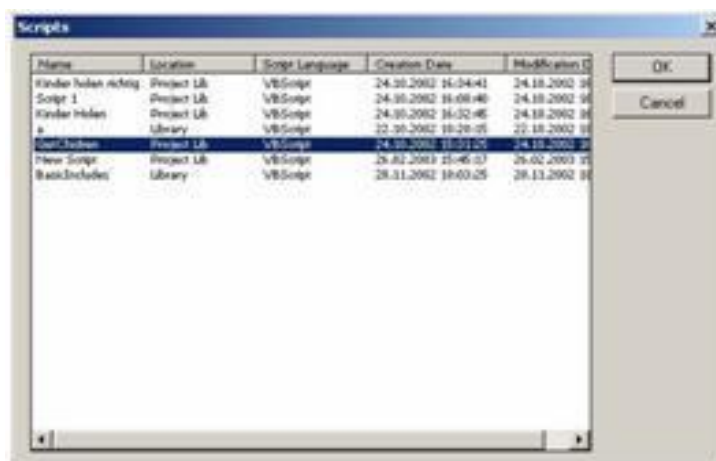


Figure 7: List of Available Scripts



### Note

*It is absolutely pointless to start any script from any component. This is the reason why option “execute Script” is deactivate by default.*

## 3.3.2 Script in the Interactive Mode

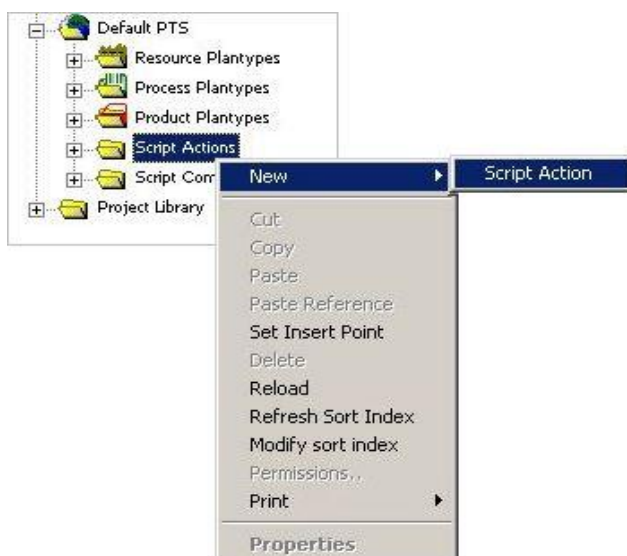
Up to now, scripts have been started by the “Execute script” entry in the context menu of a component. This method should not be regarded as the standard. If you have many scripts, searching for the right one is time-consuming and prone to error. Therefore, you are recommended to preferably use one of the methods described below for starting a script instead of the method described previously.

## 3.3.3 Starting Scripts Automatically by using Script Actions

In addition to the “Execute script” call in the context menu of a component, there are further options to start scripts in the context menu.

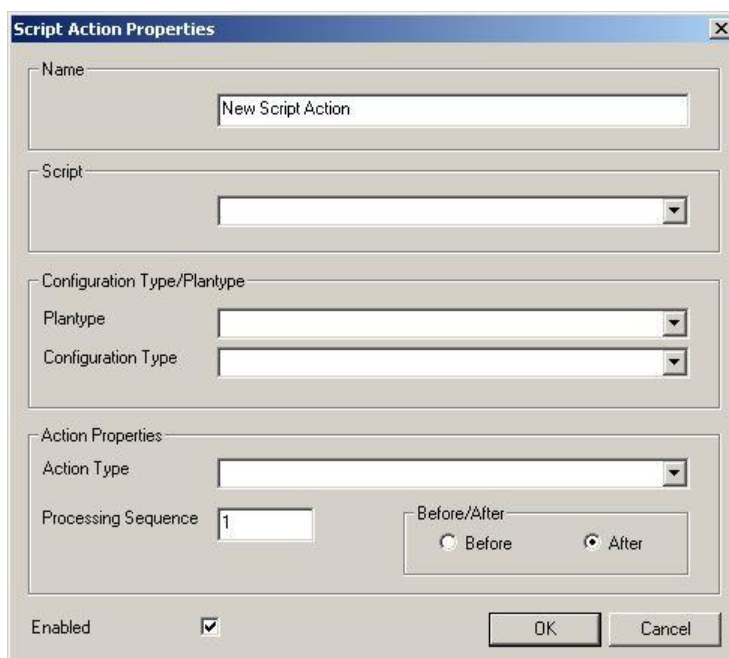
You have already been shown the Script Actions folder in section [Scripts, VBA Macros, and Script Actions](#). This is the folder that you also use to start scripts automatically. During the automatic start of script actions, scripts are automatically started together with certain actions. Actions include the creation, the copying, and moving of a new component (*Please refer to the Table 3*). When executing script actions, either the entered type or the plantype is taken into account. These entries are made when you create a script action (*Please refer to the Figure 9*). Proceed as follows:

- 1) Select the “Script Actions” folder in the project plantype set.
- 2) Create a new script action using the **New** entry in the context menu.



**Figure 8: New Script Action**


- 3) Doubleclick the newly created script action to open the dialog. Enter a name in the open dialog.
- 4) Then use the **Script < VBA Macro** menu item to select the script that you want to execute a script action for. *Please refer to the [Figure 9](#).*
- 5) Under **Type < Plantype** enter, For example, ergoproject in the entry field configuration type. Click **OK** to save and exit the script action. You can now start this script in the project node context menu.



**Figure 9: Script Action Properties**

If you have entered a plantype (i.e. a subassembly) instead of a configuration type, the script is not started in the project node context menu but in the subassembly context menu). *Please refer to the [Table 2](#).*

Table 2: Table with Entry Parameters with Script Actions

Attribute	Value	Comment
Script	Script for executing	Here you select the name of the script that you want to assign an action to.
Plantype	Plantype which the script is to be executed for.	Specify either the plantype or the type here.
Type	Type which the script is to be executed for.	Here, you specify theType if there is one type for a plantype selection. A type might quite possibly have been assigned several plantypes. Or if a script action is to be executed on an object which is not assigned a plantype.
Action Type	Information on the script action type	This type has to be compatible with the included script. For example, "Copy" has to correspond to a "sa_copy" function in the script.
Before/After	Information on whether the script is to be executed before or after the action.	If you select "Before", the script can block the current action by the return of "False". <b>Note:</b>  Some actions can only steer a "Before" or an "After" script action, For example, sa_delete.
Enabled	Deactivation or activation of the automatic script execution	From PE 5.9 only.
Processing sequence	An integer that specifies the sequence. 1 is the default value.	If several script actions are specified for one action and plantype (type), the order of the scripts to be executed can be specified here, provided that the scripts are not meant to be executed in random order.

To start a script action, you need to have the corresponding function. Actions triggering script actions are described below (*Please refer to the [Table 3](#)*)

Table 3: Table for Actions

Event	Actions in order of Invocation
Creating a new component ( <b>New ...</b> ) in PPR Navigator and PPRClients	NewChild/Before AddChild/after SetAttribute/before SetAttribute/after New/After
<b>Copying</b> component(s)	Copy/before SetAttribute/before ** SetAttribute/after ** Copy/After ** If "properties on copy" flag is enabled.
<b>Linking</b> component(s)	Link/before Link/after
<b>Moving</b> component(s)	Move/before Move/after

Event	Actions in order of Invocation
<b>Deleting reference</b> (subcompitem/relationship)	RemoveChild/before RemoveChild/after
<b>Deleting components</b> (ergocompbase/ergoitem / Others)	Delete/Before <b>Delete/After (PE5.15SP3)</b>
On <b>initialization</b> of a <b>properties dialog</b>	InitProperties/After
On changing the value of a combo box or list box control within a properties dialog	ChangeProperties/After
<b>Changing</b> properties	SetAttribute/before SetAttribute/after
Creating a <b>new version</b> or trying out a new version	Neue Version/before NewVersion/after
<b>Changing the planning state</b> or checking a version	SetPlanningState/before SetPlanningState/after
With <b>opening of a project</b>	OpenProject/after
Before <b>closing a project</b>	CloseProject/before
On launching a balancing sheet.	StartBalancing/After
On dropping processes to a balancing sheet.	DropProcessToBalancing/After
On <b>saving a balancing</b> sheet.	SaveBalancing/After
Before <b>showing</b> or editing a <b>graphic</b> object.	Open Graphic/Before
<b>Printing</b>	Print/before Print/after
<b>Deleting reference</b> (subcompitem/relationship/ simplereleation)	RemoveChildEx/before RemoveChildEx/after

### 3.3.4 Creating Real Script Actions

Real script actions are automatically executed with certain actions unlike Script Commands which are triggered manually. If a certain action occurs, a script which has been assigned this action by a script action is executed automatically (For example, adding a new object to a structure or linking it to another object).

Real script actions are executed each time a certain action is started. The script can either be started **before** or **after** these actions.

Real script actions differ from Script Commands-script actions with regard to another aspect. Whereas you use Script Commands-script actions to control “normal” scripts ....or VBA Macros, real script actions have to have a function which corresponds to the action that the script action is to be linked to.

#### Example

You add a new object to a structure; with the creation of the script action you have specified beforehand whether the script is to be executed **before** or **after** this procedure. The script itself has to contain the [function](#)  
[sa\\_new\(new\\_object\\_id\)](#)

Real script actions are created in the same way as Script Commands-script actions; the only difference between them is that when creating a **Properties**

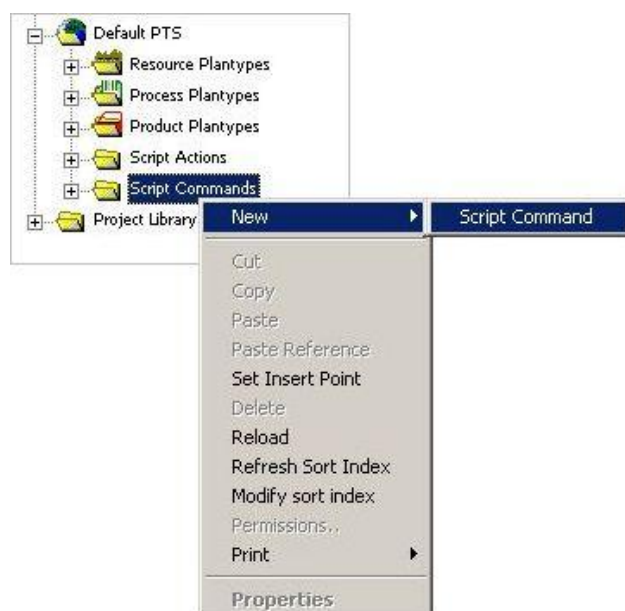


dialog, you do **not** have to select the *Popup Action Type*, but one of the script actions which are to be executed with the script. All further entries in the **Properties** dialog correspond to the entries with Script Commands-script actions. For a description of single actions which trigger script actions *Please refer to the [List of Script Actions](#)*.

### 3.3.5 Creating Script Commands

Script Commands link a selected plantype or type to a script or a VBA Macro which are executed using the Scripts or VBA Macros entry in the plantype or type context menu. This context menu entry is created due to the connection with a plantype or type in the **Script Actions** Properties dialog. Proceed as follows:

- 1) Select the “Script Commands” folder in the project plantype set.
- 2) Create a new script action using the **New** entry in the context menu.

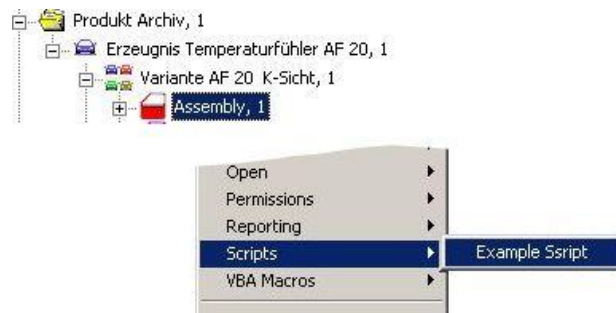


**Figure 10: New Script Action**

- 3) Under **Type < Plantype** enter, for example, ergoproject in the entry field configuration type. Click **OK** to save and quit the script action. You can now start this script in the project node context menu.

If a plantype (i.e. group of work places) was entered instead of a configuration type the script does not start via the context menu of the project node, rather via the context menu of a group of work places. *Please refer to the [Figure 11](#)*.





**Figure 11: Starting Script by Context Menu on Production Node**

For further explanations of the entry parameters, [Please refer to the Description of Entry Parameters for Script Commands Actions.](#)



### Caution


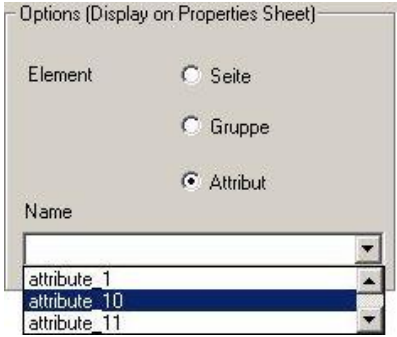
*Script Commands are only to be executed by administrators or colleagues who have the same administration rights.*

**Figure 12: Creating Script Commands – Properties Dialog**

### Description of Entry Parameters for Script Commands Actions

**Table 4: Script Commands Entry Parameter**

Attribute	Value	Comment
Name	Name	
Prompt	Prompt	This sequence of signs is displayed as a context menu entry.
Script	Script for executing	Here, you select the name of the script that you want to assign an action to.
VBA Macro	VBA Macro for	Here, you select the name of the VBA Macro that you

Attribute	Value	Comment
	executing	want to assign an action to.
Plantype	Plantype which the script is to be executed for.	Specify either the plantype or the type here.
Configuration Type	Type which the script is to be executed for.	<p>Here, you specify theType if there is one type for a plantype selection. Several plantypes might quite possibly have been assigned to one type.</p> <p>Or if a script action is to be executed on an object which is not assigned a plantype.</p> <p><b>Note:</b>  You must specify a “leaf” type in an inheritance hierarchy. The query does not cover base types like “dodefaultimpl”, although all types can be selected.</p>
Visible	<p>Activates or deactivates the entry of a script or VBA Macros in a dialog or context menu.</p> <p>Display in context menu</p> <p>Display in properties dialog</p>	<p>► If no checkbox is activated, no script call-up is possible via the context menu or properties dialog, even if a script assignment was created. The same results from deactivating the checkbox <i>Visible</i>, regardless if the other two checkboxes are activated.</p> <p>► If the checkbox Display in the context menu is activated, the script is called-up via the context menu of an object.</p> <p>► If the checkbox Display in properties dialog is activated, the script is called-up via a button in the properties dialog of an object. As of version PE 5.16SP6 the display in properties dialog differentiates between pages, groups and attributes. The type of display is configured under Options.</p> <p>Script assignments can be defined for plantypes only.</p> <p>A page, group, or attribute must be configured correspondingly so that a script can be executed on a page, group or attribute. The property <i>Can have Macro assigned</i> is found in the configuration manager. This property must be set to <b>Yes</b>. In order to assign the button, select the button position in the Position property.</p>
Display in context menu/properties dialog	<p>Page</p> <p>Group</p> <p>Attribute</p>	<p>These options define where the script action is executed. In the drop-down field below, the listed values are contingent upon the selection made. In this example the script action is executed with attribute 10.</p>  <p>If the script action is executed on a page or a group, the drop down field will offer the configured pages (and deviations) of the selected plantype.</p>
Script execution	<p>For each object</p> <p>Once for all objects</p>	<p>This setting is important for multiple selections only.</p> <p>In case multiple objects were selected in a list, this entry defines whether the scripting <i>main-method</i> is used for</p>

Attribute	Value	Comment
		all selected objects or if the scripting <i>valuation-method</i> is called up for each selected object. Of course the <i>valuation</i> method must exist in the script or else an error message is displayed. <i>Please refer to the <a href="#">The Valuation Method</a>.</i>

### Different Plan Types

If multiple objects of **different** plan types are selected in PPR Navigator list view the assigned script commands must have the same name and description and must execute the same script. If one of this rules is not fulfilled the script command is disabled in the PPR Navigator.

The referenced script(s) and/or VBA macro(s) are displayed depending on the value/format of the attribute prompt.

### 3.3.5.1 The Valuation Method

The *valuation* method is used in connection with a script assignment and the corresponding setting *Script execution*.

The setting *Script execution* offers two options:

- **For each object:** The *main method* is called-up for each selected object as a start method.
- **Once for all objects:** The *valuation-method* is called up for each selected object. The *main* method is ignored as a starting method or classified as a regular subroutine and the *valuation* method becomes the start method.
- For a script to work on both settings, it needs a *main method* and a *valuation method*.

### Example

```
Sub main(id)
    call Dialog.MessageBox("Main", "Single object_id is: " & id )
End sub

Sub valuation(ObjectIds)
    i=0
    For Each ecursiv In ObjectIds
        i=i+1
        call Dialog.MessageBox("Valuation", "Id of object(" & i & ")
is: " & ecursiv )
    Next
End sub
```

If the setting of a script assignment is set to Once for all objects and the *valuation* method is missing, then the script gets aborted with the following error message.



Figure 13: Error Message for a Missing Valuation Method

### 3.3.5.2 Executing Script in the Context Menu

The script or VBA Macro is to be executed in the context menu of the assigned plantype or type.

- 1) Select the *plantype* or *type* in the PPR-Navigator and then open the context menu.
- 2) In the context menu you get either find the Scripts (in the case of assigned scripts) or VBA Macros (in the case of assigned macros) entry.
- 3) Click the corresponding entry (i.e. script name, *Real Children*) to execute a script or a macro.

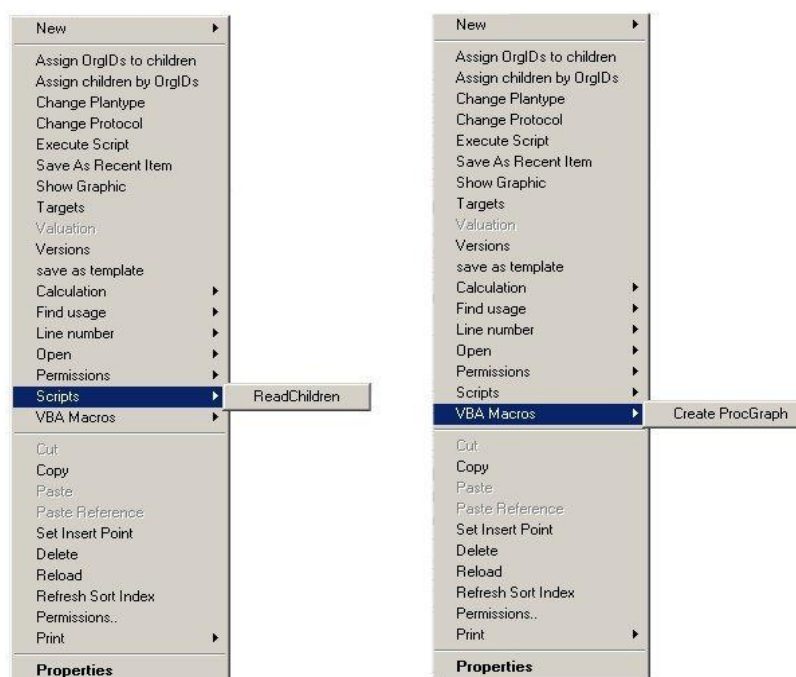
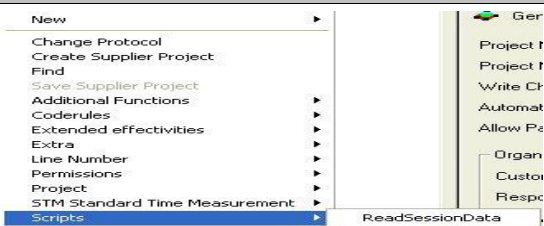
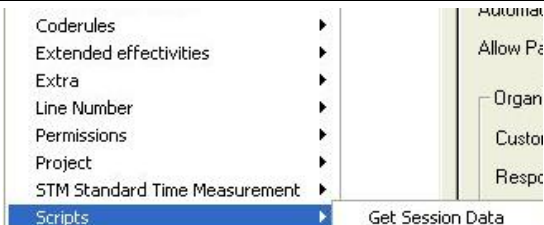
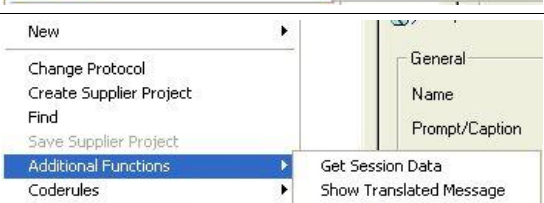


Figure 14: Executing a Script or a VBA Macro in the Context Menu

The option to open scripts via the context menu „Scripts“ has been extended, so that scripts can now be opened via an associated, function-related context menu as well.

Therefore two string values must be separated with a backslash in the name. The backslash serves as a separator. The different call-up options are summed up in the following table.

Table 5: Call Up Options

Prompt	Example	Result
Empty	(The name of the script or VBA macro is used. In this example, the script's name is "ReadSessionData".)	
A string value	Get Session Data	
A string value	Additional Functions\Get Session Data	

### 3.3.5.3 Executing a Script in the Properties Dialog of an Object

- 1) In order to call up a script or a VBA Macro in the properties dialog of an object, the property **Visible < Display in properties dialog < Page** must be activated when creating a script assignment.
- 2) Select a page on which the script is called-up.

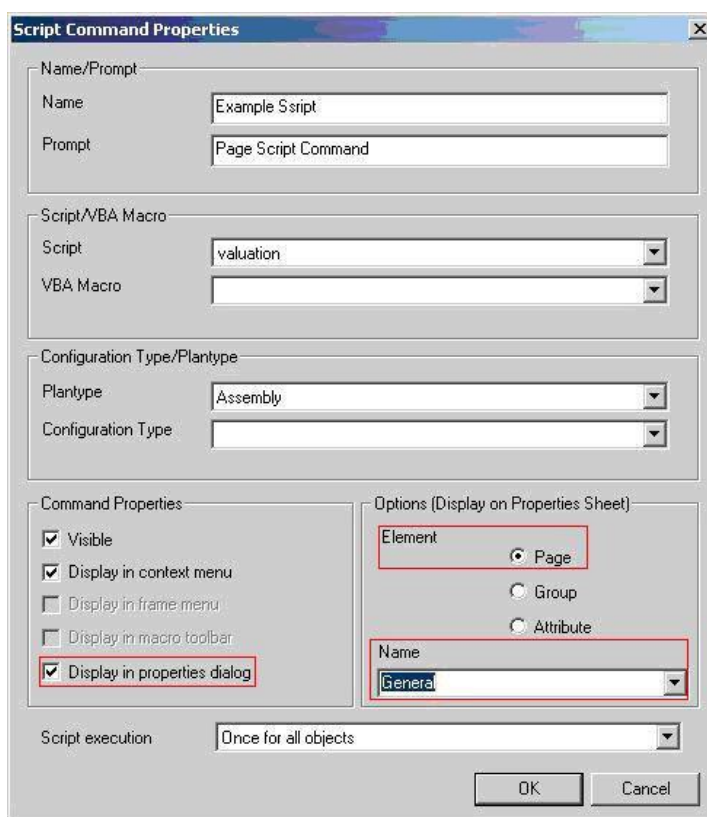
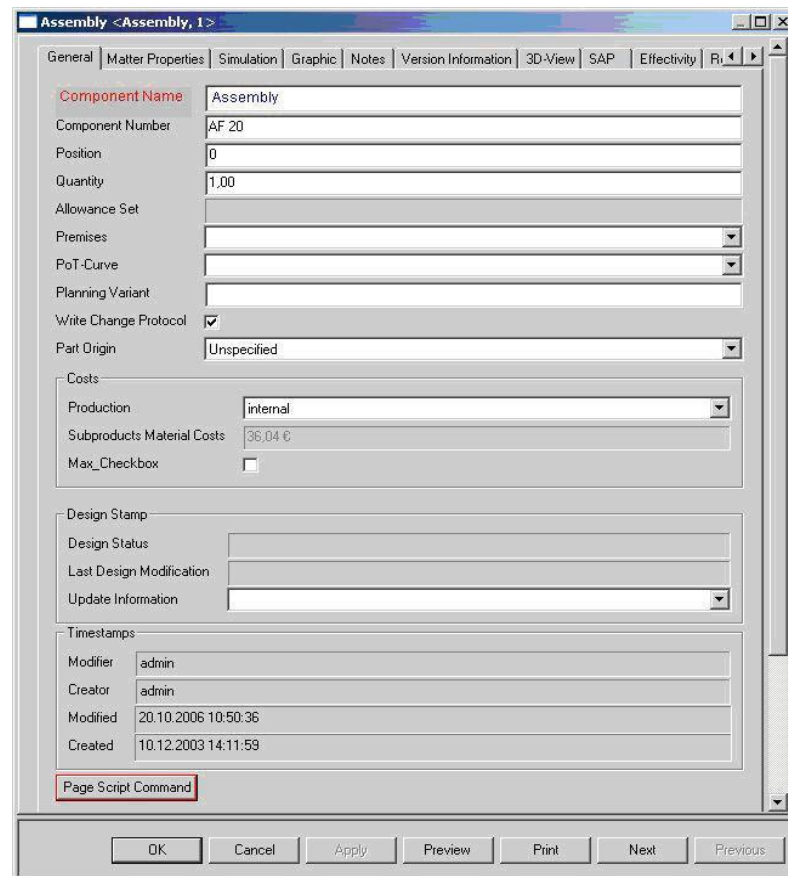


Figure 15: Script Command Properties

- 3) Select the **plantype** in the PPR Navigator and open the **Properties** dialog. The button **Page ScriptCommand** is located under the tab **General** in the **Properties** dialog. Please refer to the [Figure 16](#).
- 4) In order to execute a script or macro click the corresponding button (i.e. name of script, Page ScriptCommand). Multiple scripts can be displayed in a **Properties** dialog.

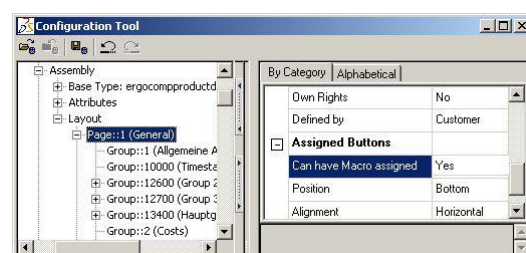


**Figure 16: Script Call-up in the Properties Dialog of an Object**

### **Example of a Script Assignment for an Attribute**

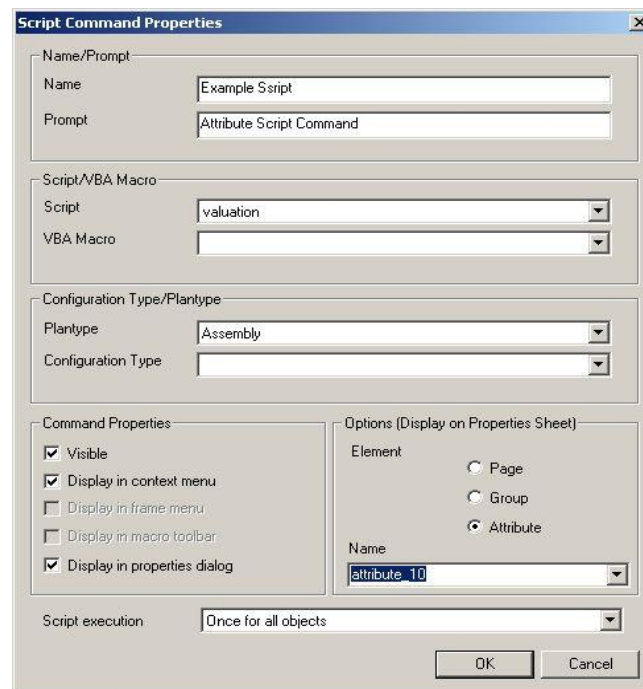
In this example a script assignment is created for the plantype Part.

#### **3.3.5.2 First the attribute is checked, to determine if it has been configured for the assignment of scripts.**



**Figure 17: Checking of Attribute**

### 3.3.5.3 The script assignment is created.



**Figure 18: Script Assignment**

- 3) An object of the plantype Part is opened to check if the script action was created correctly.



**Figure 19: Plantype Part Object**

## 3.3.6 Automatic Execution of Scripts

### 3.3.6.1 Automatic Execution of Scripts with Script Action

Please refer to the [Starting Scripts Automatically by using Script Actions](#) and [Running Scripts in Batch Mode](#).

## 3.3.7 Running Scripts in Batch Mode

Batch mode is recommend to run scripts which take a long time for execution and/or are intended to fulfill some regular maintenance tasks over night without user interaction.

Up to PE 5.13, this required to create the script object for such a mode inside Process Engineer and to retrieve the object ID of that particular script.

This kind of batch mode is still available; however, we have extended the capabilities in order to be more independent.



For PE 5.14 and later, it is possible to pass the name of a file containing the script code to be executed.

### 3.3.7.1 Batch Execution from File

Until PE5.15

### 3.3.7.2 Run the executable.

```
\\PPRClient\program\bin\DPFFrame.exe
```

### 3.3.7.3 Run with these parameters.

```
/username:[login] /userpwd:[pwd]  
/startobject:ScriptEngineLoader.ScriptEngineLoaderManager „[filepath]”[[entry  
function]][[script language]]“
```

#### Example

```
\\PPRClient\program\bin\DPFFrame.exe /username:admin /userpwd:admin  
/startobject:ScriptEngineLoader.ScriptEngineLoaderManager  
„c:\temp\myScript.txt|main|vbscript“
```

- The second and third parameters are optional. The default entry function is **main**, the default script language is **vbscript**.
- The entry function **must not have** entry parameters! This is different from standard Process Engineer scripts, in which the entry function **main** takes a single parameter, the ID of the invoking object.

#### PE5.15 and later

```
Parameters: „<USERNAME>login</USERNAME>  
<E5_PWD>pwd</E5_PWD>  
<STARTOBJECT>ScriptEngineLoader.ScriptEngineLoaderManager</START  
OBJECT> [filepath]”[[entry function]][[script language]]“
```

#### Example

```
\\PPRClient\program\bin\DPFFrame.exe  
„<USERNAME>admin</USERNAME> <E5_PWD>admin</E5_PWD>  
<STARTOBJECT>ScriptEngineLoader.ScriptEngineLoaderManager</START  
OBJECT> c:\temp\myScript.txt|main|vbscript“
```



#### Note

- The parameters **[entry function]** and **[script language]** are optional. The default entry function is **main**, the default script language is **vbscript**.
- The entry function **must not have** entry parameters! This is different from standard Process Engineer scripts, in which the entry function **main** takes a single parameter, the ID of the invoking object.

Please note that in the PE 5.14 version, the quotation marks enclose the block “**[filepath]”[[entry function]][[script language]]**”, while in the PE5.15 version the quotation marks enclose the entire command line after the program name.

- In the PE 5.14 version, the blank between the start object name and the first quotation mark is mandatory, in PE 5.15 the blank between the **</STARTOBJECT>** end tag and the filename is mandatory.



### 3.3.7.2 Batch Execution from an Object Until PE 5.15

#### 3.3.7.4 Run the executable.

```
\\PPRClient\program\bin\DPFFrame.exe
```

#### 3.3.7.5 Run with these parameters.

```
/username:[login] /userpwd:[pwd]  
/startobject:ScriptEngineLoader.ScriptEngineLoaderManager „[script object  
ID]”[main entry object ID]“
```

##### Example

```
\\PPRClient\program\bin\DPFFrame.exe /username:admin /userpwd:admin  
/startobject:ScriptEngineLoader.ScriptEngineLoaderManager „$id$(0:0-  
7704#0, 339)|$id$(0:0-7493#0, 168)“
```

##### PE5.15 and later

```
Parameters: „<USERNAME>login</USERNAME>  
<E5_PWD>pwd</E5_PWD>  
<STARTOBJECT>ScriptEngineLoader.ScriptEngineLoaderManager</START  
OBJECT> [script object ID]”[main entry object ID]“
```

##### Example

```
\\PPRClient\program\bin\DPFFrame.exe  
„<USERNAME>admin</USERNAME> <E5_PWD>admin</E5_PWD>  
<STARTOBJECT>ScriptEngineLoader.ScriptEngineLoaderManager</START  
OBJECT> $id$(0:0-7704#0, 339)|$id$(0:0-7493#0, 168)“
```



##### Note

► The parameter [main entry object ID] is optional. If a main entry object ID is required, then concatenate this ID to the script object ID using a “|” (pipe) character.

Otherwise pass the script object ID alone (without the pipe character).

► Please note that in the PE 5.14 version, the quotation marks enclose the block “[script object ID]”[main entry object ID]”, while in the PE5.15 version the quotation marks enclose the entire command line after the program name.

You can retrieve an object ID from virtually any node in the PPR Navigator using the following script.

```
Sub main(id)  
    val = InputBox(“Object ID is”, “Display ID”, id)  
end sub
```



Figure 20: Display ID

## 3.4 Appearance of Scripts in the Process Engineer

Now, we have a look at the possibilities that scripting offers within the DELMIA Process Engineer.

You just need to know one basic thing:

Each script you write needs to be entered into method which serves as a starting point for the execution. At present, there are several methods serving as a starting point, i.e. the “main” method, for the different script action functions and the starting method in VBA.

“Normal” scripts at least contain the following structure:

```
Sub main (object_id)
  ..
end Sub
```

It is up to you to program the contents within this structure. You can write your complete code within this method; for certain complexity you are recommended to structure the program, i.e. to divide it into subprograms; i.e. further submethods are called from the “main”-method.

You might have noticed the **object\_id** transfer parameter in this code fragment.

### Introduction to object\_id

Almost every object you can view in the tree or list view of DELMIA Process Engineer is persistent, i.e. saved permanently in your database which is the PPR Hub; the object is identified by an unambiguous ID which is the object ID. The object ID of the object can automatically be transferred to the script when you start the action on when you select the “Execute script” action in the context menu. It is not important whether or not the object ID is used in the script. The object ID is a string which is recognizable by „\$id\$“ beginning. The object ID of the object can therefore be transferred since it is often necessary to find out attributes of this component or assigned bill of materials entries, relations, and other things. Example of a command line using the **Object\_id**:

```
Name = data.GetAttributebyId(object_id, "name")
```



### Note

*There is a further entry into methods in subsequent versions of the DELMIA Process Engineer.*

In addition to the **main** routine, there are two other methods:

### Procedure

A procedure is a named order of commands, which are executed as a unit. Function and Sub, for example, are procedure types. The name of a procedure is always defined on the module level. The entire executable code has to be included in a procedure. Procedures cannot be inserted into other procedures; they can, however, be called by other procedures. A procedure does not have a return value.

### Functions

A function has the same structure as a procedure. However, unlike the procedure, the function has a return value.

Now it is obvious why the initial example:

```
MsgBox „Hello, this is an output box“, „My first script“
```

generated an error message.

If you execute the script in the **main** procedure, you does not get an error message.

```
Sub main (object_id)
  MsgBox "Hello, this is an output box", , "My first script"
End Sub
```

### Introduction to Variable

A variable is a named storage position which contains data that can be changed during program execution. Each variable has a name which unambiguously identifies this variable in its range of validity. In addition, a data type can be specified.

Variable names have to begin with a letter; they must be unambiguous within the range of validity and must not have more than 255 characters. Variable names must not contain a dot or a type ID.

## 3.4.1 Error Handling

It would be a miracle, of course, if everything succeeded at the first go, but generally, several error messages occur especially for larger scripts.

Possible error types:

- Syntax errors in the script
- Runtime errors
- Logical errors (typical example: Division by zero)
- Invalid parameters with methods of foreign components (i.e. file, database access)
- Invalid parameters with methods of the DELMIA Process Engineer components (data, query, grid...) with calls of unknown methods of existing objects.

**Syntax errors** are recognized by the script-engine before program execution and quit by a corresponding error message.

**Run-time errors** are difficult to deal with. The following applies in general: If there is no explicit error handling in a script, the script is cancelled after an error occurs.

This means that some data might possibly have been changed as well as the initial state. Canceling is not possible unless all changes are logged so that they could be "actively" reset.

### Example

```
REM script with error handling for access to the attributes of a
component.
` The first call up is done with a false attribute name ("name-
mea") and leads to a
` respective error report.
` The second call up is correct (Attribute name "name"). The me-
thod OnErrorResumeNext
` prohibits the cancellation of the script when an error comes
up!
` the error object "Err" with an error will be "filled" and must
afterwards be with "Err.Clear"
` cleaned up again.
Sub change_attribute(object_id, attributename, value)
```

```

On Error Resume Next
call data.SetAttributebyId (object_id, attributename, value)
if Err.Number <> 0 then
MsgBox(Err.Description)
Err.Clear
else
MsgBox("Execution correct")
end if
end sub

Sub main (object_id)
  REM call up with a false attribute
  call change_attribute(object_id, "namea", "Test")
  REM call up with a correct attribute
  call change_attribute(object_id, "name", "Test")
end Sub

```



### Caution

*The "" string is used as a cancel string for GetFirstChild/GetNextChild.*

## 3.4.2 Recursions

Complete tree structures has to be passed through very often. To do this, you can call script methods recursively. It is also possible to call the **main** method recursively as shown in the example below: However, the clear structure of such scripts, particularly longer ones, cannot be retained.

### Example

```

REM general (rekursiv)
Sub main(parent_id)
  child_id = data.GetFirstChild(parent_id, "nodes")
  If child_id <> "" Then

    Do While child_id <> ""

      main(child_id)
      ..command ...
      child_id = data.GetNextChild(parent_id, "nodes")
    Loop
  End If
End Sub

```

In general, with such calls and loops it is important to remember the cancel criteria. Otherwise, there you get infinite loops.

```

REM Cumulate Invest (one level)
Sub main(parent_id)
wert = 0.0
child_id = data.GetFirstChild(parent_id, "nodes")
If child_id <> "" Then
  Do While child_id <> ""
    wert = wert + data.GetAttributebyId(child_id, "dummyinvest")
    child_id = data.GetNextChild(parent_id, "nodes")
  Loop
  data.SetAttributebyId parent_id, "calc_estim_invest", wert
End If
End Sub

REM Cumulate Mass (recursive)
Sub main(parent_id)
wert = 0.0 child_id = data.GetFirstChild(parent_id, "nodes")
If child_id <> "" Then

```

```

Do While child_id <> ""
  main(child_id)
wert= wert + data.GetAttributebyId(child_id, _
  "mass")*data.GetAttributebyId(child_id, "quantity")
  child_id = data.GetNextChild(parent_id, "nodes")
Loop
  data.SetAttributebyId parent_id, "mass", wert
End If
End Sub

```

To pass through the list several times, the iterator of a certain child list (which is the run pointer) is reset to ScriptItemData by means of the ResetIterator.

You can also search for pointer attributes or set pointer attributes. Search for all components of the „ergocomplantdefault“ type of the current project.

*REM search for all resources (type „ergocomplantdefaults“) of the REM project*

**Sub main (id)**

**project\_id=data.getattributebyid(id,“ergoproject“)**

```

query.ResetSearch
query.SetQuery "ergocomplantdefault","m_pErgoProject","=", _
project_id
result_id=query.GetFirstResult
Do while result_id <> ""
  wert = data.GetAttributebyId (result_id, "name")
  MsgBox(wert)
  result_id=query.GetNextResult
Loop

```

**end Sub**

This script has to be called in the project context menu because the **project\_id** of the project is used as a criterion for comparison.

If it is not called on the project, then this ID has to be provided in a different way.

Objects (types, plantypes) can have attributes which cannot be called by a configured name (i.e. „name“) because they are normally only used internally. In the example above, however, it is crucial to specify the project in which the search is to be executed. Therefore, the „m\_pErgoProject“ pointer attribute must be used as the attribute name here.



### Note

*VBScript is complicated regarding the placement of brackets. In the examples, brackets sometimes be missing if a method has several parameters. **This is mandatory!***

For placement of brackets, proceed as follows:

- 1) If a method has got a return value, the parameters has to be put in brackets: `value = data.GetAttributebyId(child_id, „dummyinvest“)`
- 2) If the method does not have any return value, fr example, `data.SetAttributebyId parent_id, „calc_estim_invest“`, value brackets **must not** be used.

If that is too complicated, you can use the **call** command. Thus, the last code line can also be written as follows:

```
data.SetAttributebyId parent_id, "calc_estim_invest", value
```

**Further Examples****Example**

```

REM WriteChildrenNamesToTxt ( recursive)
Sub main(parent_id)
  Set fso = CreateObject("Scripting.FileSystemObject")
  Set ts = fso.CreateTextFile ("c:\test.txt", ForWriting, True)
  ReadChildren parent_id, ts
  ts.Close
End Sub
Sub ReadChildren(parent_id, ts_in)
  name=""
  child_id = data.GetFirstChild(parent_id, "nodes")
  If child_id <> "" Then
    Do While child_id <> ""
      ReadChildren child_id, ts_in
      name = data.GetAttributebyId(child_id, "name")
      ts_in.WriteLine(name)
      child_id = data.GetNextChild(parent_id, "nodes")
    Loop
  End If
End Sub

```

**Example**

```

REM example for using a Dictionary Object
REM creates a file C:\test.txt, in which the shortnames and
REM attribute 1 of those processes are listed, by which REM the
REM combination of both is not singularly definite.
Sub checkProcess( parent_id, ts, d )
  Value = ""
  value = data.GetAttributebyId(parent_id, "nameshort")
  value2 = ""
  value2 = data.GetAttributebyId(parent_id, "attribute_1")
  wertkomb=wert+wert2
  If (d.exists (wertkomb) ) Then
    text = "Prozess already exists: "+wertkomb
    ts.WriteLine(text)
  Else
    d.Add wertkomb, parent_id
  End If
  child_id = data.GetFirstChild(parent_id, "nodes")
  If child_id <> "" Then
    Do While child_id <> ""
      Call checkProcess(child_id, ts, d)
      child_id = data.GetNextChild(parent_id, "nodes")
    Loop
  End If
End Sub

```

**Sub main(parent\_id)**

```

Set fso = CreateObject("Scripting.FileSystemObject")
Set ts = fso.CreateTextFile("c:\test.txt", True)
Set d = CreateObject("Scripting.Dictionary")
Call checkProcess ( parent_id, ts, d )
ts.close

```

**End Sub****3.4.3 Using Scripts in other Scripts**

By introducing scripts and script actions in the system library it is possible and even intended for other scripts to be used in the actual scriptcode. This subject can best be explained by means of a simple example:

To access data it is necessary to call the right objects. The following call is used for this purpose:

```
base_id = Data.GetAttributebyId(sci_id, „ergocompbase“)
```

This call function is to be made easier and reusable. The following steps are necessary:

- 1) Create a script in the system library.
- 2) Enter a name, for example, „**BasicIncludes**“.
- 3) Enter the following scriptcode:

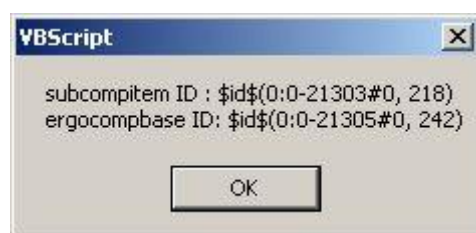
```
function GetBase(id)
GetBase = Data.GetAttributebyId(id, "ergocompbase")
end function
```

- 4) Save the script.
- 5) Create a script in the project library.
- 6) Enter a name for the script.
- 7) Enter the following scriptcode:

```
#include<BasicIncludes>

sub main(id)
base_id = GetBase(id)
MsgBox("subcompitem ID : " & id & vbCRLF & "ergocompbase ID: " &
base_id)
end Sub
```

- 8) Save the script.
- 9) You can now execute this script on each node except for the project node. As a result, the following message appears (*Please refer to the [Figure 21](#)*).



**Figure 21: Example of a Message with Usage of Scripts in a Script**

### Using another Script

The **include** instruction in your „main“ script forces the program to search for a script the name of which is between the „< >“ (square brackets). The search begins within the project library which the “main” script belongs to. If there is no match, the system library is searched for a script. If the script is found, the script interpreter transmits the following:

```
function GetBase(id)
  GetBase = Data.GetAttributebyId(id, "ergocompbase")
end function

Sub main (id)
```

```
base_id = GetBase(id)
MsgBox("ID of subcompitem: " & id & vbCRLF & "ID of ergocomp-
base: " & base_id)
End Sub
```

### Scripts with Identical Names in the Project Library and System Library

The script in the project library has priority. The script in the system library are ignored.

However, by writing the following code, you can force the DELMIA Process Engineer to use the library script:

```
include<Lib/BasicIncludes>
```

### Script accidentally used Several Times

This is a theoretic question as each script is only used once.

## 3.4.4 Logging of Script Actions

Use the **Settings** dialog in the **Tools** menu to activate the log functions for scripts. *Please refer to the [Figure 22](#).*

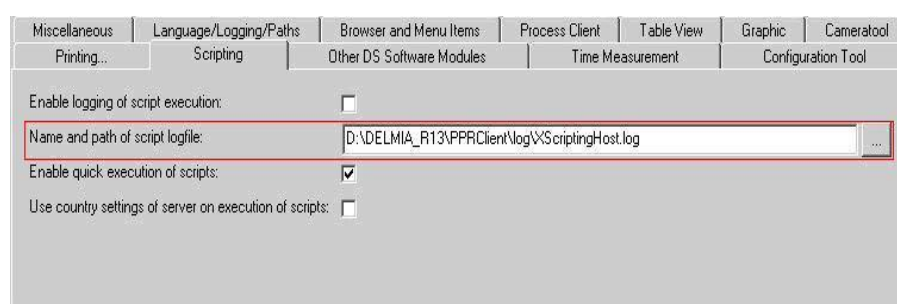


Figure 22: Enabling Log Functions – Tools – Settings



### Caution

*The execution of a script action is decelerated by activating the log function.*

Checkmark **Enable logging of script execution** to activate the log functions.

If the log function is enabled, the log file is updated permanently. It is therefore recommended to archive the log file periodically. After archiving, the log file can be deleted from the directory.

## 3.5 Using VBA Host for Programming

### 3.5.1 Introduction

Visual Basic Script is an easy-to-learn programming language that can quickly be integrated in the user programs of the Process Engineer. It is an opportunity to get to know and to apply the script functions. However, the **VBScript** programming language is not as powerful as other programming languages. The **Visual Basic for Applications** programming language from version PE 5.10 and later is used to describe complex themes in their own development environment by means of scripts in the Process Engineer.



The integrated VBA in the Process Engineer provides its own comfortable development environment for writing scripts. This environment contains:

- Syntax highlights
- Error correction (debugging)
- Intelligent function extensions such as the automatic display of further program steps during the writing of scripts.

VBA Programs are run within the started application and access the object model of the application.

The following descriptions in this chapter are based upon the VBA Environment which you are assumed to be familiar with (see Office applications). The VBA Environment (IDE) is thus explained so that you can understand the single sections.

### 3.5.2 Creating VBA Scripts

VBA Projects are created in the project library. A binary object, a so-called Binary Large Object (BLOB) is simultaneously created in the background for the information on the VBA Projects entered in the **Properties** dialog. In this binary object, the structure of all reference data of the VBA Projects is saved



**Figure 23: Creating VBA Scripts – New Context Menu**

#### **To Create VA Scripts**

- 1) Open the project library.
- 2) Select the VBA Projects folder in the project library. Select **VBA Project** in the context menu.
- 3) Enter the name and number for the script identification in the **Properties** dialog. Click **OK** to create the VBA Project.

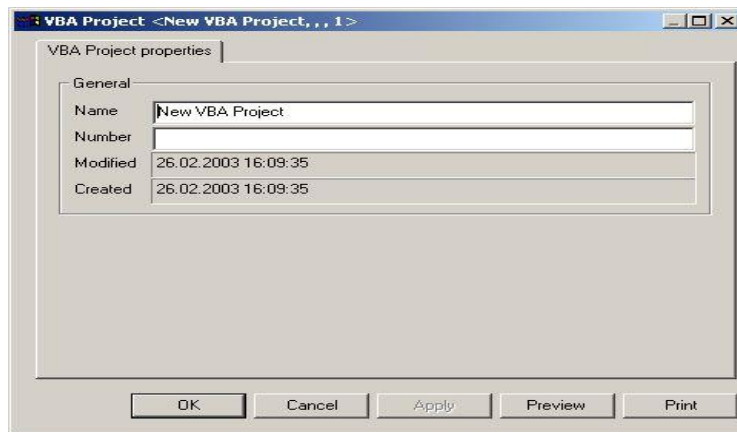


Figure 24: VBA Project Properties Dialog

### 3.5.2.1 Opening VBA Environment (IDE)

Scripts are written in the VBA Environment. The VBA Environment is opened in the context menu of a VBA Project created in the project library. In this development environment, macro codes, classes, modules, and resources are developed.




Figure 25: Opening VBA Environment – Context Menu

#### To Open VBA Environment

- 1) Open the project library.
- 2) Open VBA Projects folder in the project library. Select a **VBA Project**.
- 3) Open the context menu on the selected VBA Project. Click **Open VBA IDE** in the context menu. The VBA Environment gets opened. *Please refer to the [Figure 26](#).*

### 3.5.2.2 VBA Environment View

As an example, the [Figure 26](#) shows a VBA Environment opened in the VBA Project in the Process Engineer. In the tree structure of the project (left side) there are methods for creating scripts (XscriptItem-objects), with which you are already familiar from the Process Engineer, are displayed.

- 1) Click  icon to leave the development environment and return to the Process Engineer.

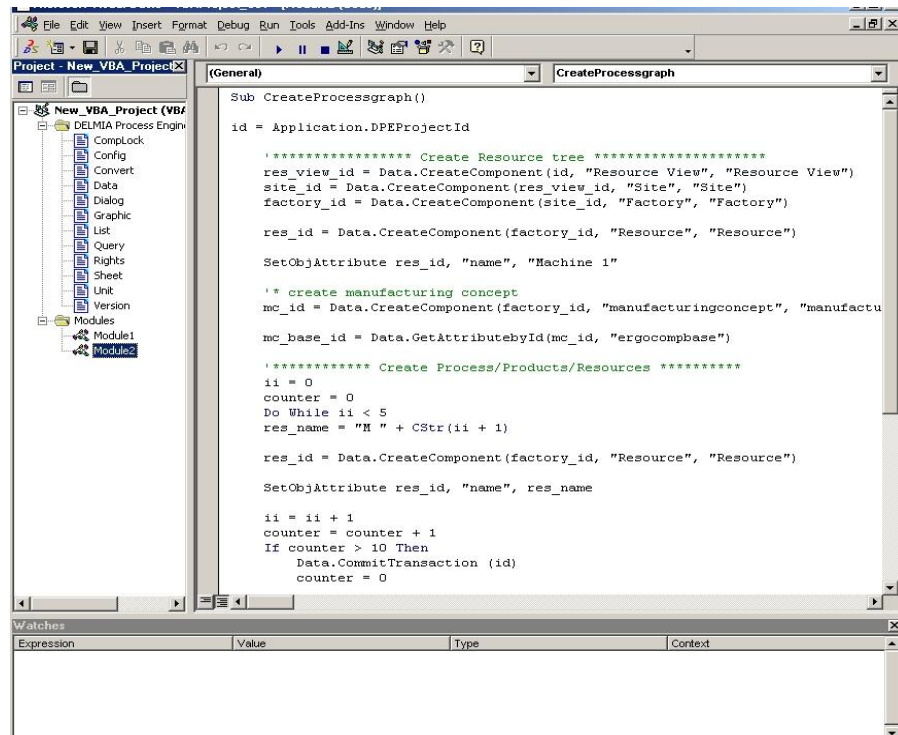
**Example**

Figure 26: Example of a VBA Environment

**3.5.2.3 Executing VBA Macros**

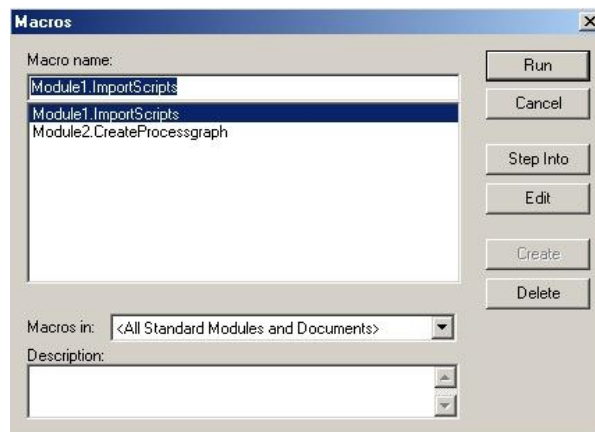
VBA Macros are executed in the project library. Use **Create** button to create and execute a new macro in the VBA Environment. Use **Edit** button to edit macros that have been created.



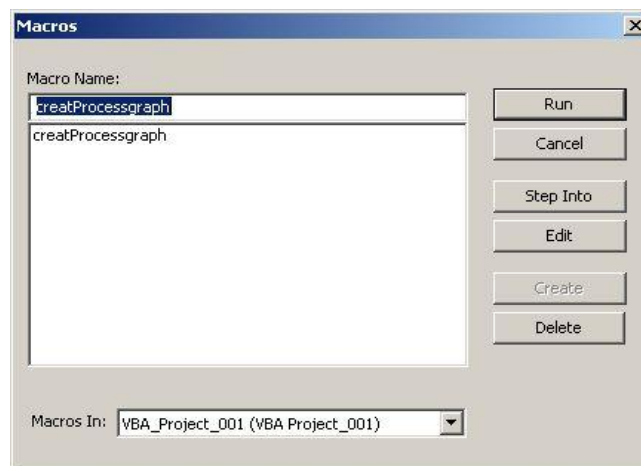
Figure 27: Opening VBA Macros in the Context Menu

**To Execute VBA Macros**

- 1) Open the project library.
- 2) Open the VBA Projects folder in the project library. Select a VBA Project.
- 3) Open the context menu on the selected VBA Project. Click **VBA Macros** in the context menu.
- 4) In the **Macros** dialog, you can select created macros for the execution project-related and project-wide using **Macros in**. Please refer to the [Figure 28](#).

**Example****Figure 28: Macros Dialog – Project-Wide Selection**

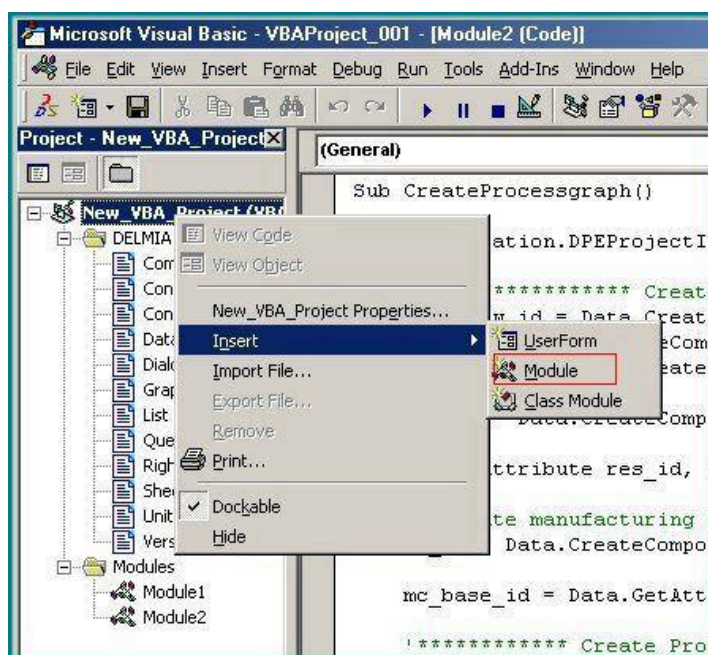
- 5) Select a macro and click **Run** button to execute a macro. The macro gets executed. For a project-related selection, *Please refer to the [Figure 29](#).*

**Example****Figure 29: Macros Dialog – Project-Related Selection**

### 3.5.3 Changing Scripts into VBA Macros

VBA Scripts are edited in the VBA Environment. To edit or convert an existing writing code, a VBA Project must have been created and the VBA Environment must be opened.

The following example shows the proceeding.



**Figure 30: Opening the Script Editor**

- 1) Select the project in the tree structure (a newly created project is shown in the picture, *Please refer to the [Figure 30](#)*.
- 2) Open the context menu and click **Module**.
- 3) Write the script in the Script Editor.
- 4) You have to write the entries according to the structure shown in the example in order for the script to be recognized as a macro. *Please refer to the [Table 6](#)*.

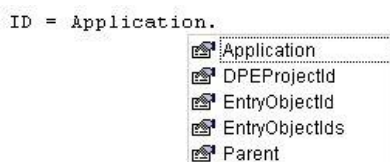
The right window shows the structure (blue letters) for the conversion of the script into a VBA Macro. *Please refer to the [Table 6](#)*.

**Table 6: Conversion of a Script into a Macro**

<pre> REM Script code sub main(entry_id) name = Data.GetAttributeById(entry_id, "name") MsgBox(name) end sub </pre>	<pre> REM VBA macro code sub DisplayName() entry_id = Application.EntryObjectId name = Data.GetAttributeById(entry_id, "name") MsgBox(name) end sub </pre>
---	--

### 3.5.3.1 VBA Object Application

Each time you write "Application" followed by a dot, the VBA IDE suggests a property of the VBA Application object for selection.



**Figure 31: VBA Application Property**

For a description of the properties, *Please refer to the [Table 7](#).*

Corresponding to the terms described in the table, these properties are dynamically assigned to a node by means of the object ID. The assigned node is the node on which the VBA Macro is executed.

**Table 7: Description of the VBA Object Properties**

VBA Object Properties	Description	
EntryObjectId	ID of the selected object	
DPEProjectId	Project ID of the project which the object belongs to (current transaction).	
EntryObjectIds	Object is basic type	Contents of the field
	subcompitem	EntryObjectIds(0) = ID of subcompitem EntryObjectIds(1) = ID of referenced ergocompbase
	relationship	EntryObjectIds(0) = ID of relationship EntryObjectIds(1) = ID of referenced relationobject
	others	EntryObjectIds(0) = object ID EntryObjectIds(1) = object ID

### 3.5.4 Error Handling with VBA Macros

The integrated debugger is one of the most important features of the VBA Environment (IDE). This debugger helps with the development of macrocodes. As far as the initial object ID is considered, the following facts need to be noted.

As already mentioned, the EntryObjectID is the ID of the selected object.

If you execute your script in the VBA Environment for test purposes, you always get the Entry ObjectID of the VBA Project. But this ID rarely corresponds to the required ID.

#### Example

```
REM VBA Macro Code
REM EntryObjectID
sub DisplayName()
    entry_id = "$id$(0:0-9201#0, 218)"
    name = Data.GetAttributebyId(entry_id, "name")
    MsgBox(name)
end sub
The script returns the ID of the object node in the PPR-
Navigator which it was executed on.
```

```
Sub main(id)
  val = InputBox("Object ID is", "Display ID", id)
end sub
```

## 4. List of Script Actions

The [Table 8](#) depicts the different types of script actions methods.

**Table 8: List of Script Actions Methods**

Script Action	Script Function to Implement	Event	Action Finder Checks ...*	Before	After
<a href="#">NewChild</a>	sa_newchild	On creating a new component.	Parent object	X	-
<a href="#">AddChild</a>	sa_addchild	Adding a component to a childlist.	Parent object	-	X
<a href="#">SetAttribute</a>	sa_setattribute	Changing attributes of a component.	Object	X	X
<a href="#">ScriptActionNew</a>	sa_new	After a new component has been created.	Object	-	X
<a href="#">Copy</a>	sa_copy	Copying components (single and multiselect) to another parent component.	Parent object	X	X
<a href="#">Link</a>	sa_link	Linking components (single and multiselect) to another parent component.	Parent object	X	X
<a href="#">Move</a>	sa_move	Moving components (single and multiselect) from one to another parent component.	Parent object	X	X
<a href="#">Delete</a>	sa_delete	Deleting a component (from the project library).	Object	X	X
<a href="#">RemoveChild</a>	sa_removechild	Deleting a link (subcompitem, relationship).	Parent object	X	X
<a href="#">ScriptActionInitProperties</a>	sa_initproperties	After standard initialization of the controls of a properties dialog (but before actually opening it)	Object	-	X
<a href="#">Change Properties</a>	sa_changeproperties	After changing the value of a combo box or list box control in a properties dialog	Object	-	X



Script Action	Script Function to Implement	Event	Action Finder Checks ...*	Before	After
ScriptActionNew Version	sa_newversion	Creating or checking out a new version.	Object	X	X
ScriptActionSet PlanningState	sa_setplanningstate	Setting the planning state or checking in a component.	Object	X	X
ScriptActionPrint	sa_print	Before and after starting a print job.	Object	X	X
ScriptActionPrint list	sa_printlist	Before and after starting a list print job.	Object	X	X
ScriptActionOpenProject	sa_openproject	After opening a project.	Object	-	X
ScriptActionCloseProject	sa_closeproject	Before closing a project.	Object	X	-
StartBalancing	sa_bal_start	After launching a balancing sheet.	Object	-	X
SaveBalancing	sa_bal_end	After saving a balancing sheet.	Object	-	X
ScriptActionDropProcessToBalancing	sa_bal_drop_proc	After dropping processes to a balancing sheet.	Object	-	X
OpenGraphic	sa_opengraphic	Before showing or opening a graphic object.	Object	X	-
RemoveChildEx	sa_removechild ex	Deleting a link (subcompitm, Relationship, simplere relation)	Parent object	X	X
Select Tree Item	sa_selecttreeitem	After an object is selected.	Object	-	X
Expand Tree Item	sa_expandtreeitem	After an object is expanded.	Object	-	X

\* Object for which the script action finder checks, if a script action exists. This is either the parent object (child creating, adding, removing, copying, linking, moving) or the object itself (set attributes, delete, new, new version, set planning state).

## 4.1 Script Actions

### 4.1.1 ScriptActionNewChild

#### Description

The associated script is executed whenever the user is going to add a new child by selecting **New** in the context menu. The parent\_id, the childname(plantype name or typename) and the childlistname are passed to the script. The script can terminate the event by returning 0, False, vbFalse (VBScript), or false(JScript).

#### Script Entry Function

**function sa\_newchild(parent\_id, childname, childlistname)**

Parameters	Description
parent_id	A string containing the ID of the parent object.
childname	<ul style="list-style-type: none"> <li>•Up to PE 5.13: A string containing either the name of the plantype or the name of the type is created.</li> <li>•PE 5.14 and later: A string containing either the GUID of the plantype or the name of the type is created.</li> </ul>
childlistname	<p>A string containing the name of the childlist to which the child should be appended.</p> <ul style="list-style-type: none"> <li>•Up to PE 5.13: For PPR components either "nodes" or the name of the plantype, for other types(relationships) the configured childlistname.</li> <li>•PE 5.14 and later: For PPR components either empty(!) or "nodes", for other types(relationships) the configured childlistname.</li> </ul>

#### Script Action Example

#### Example

```
function sa_newchild(parent_id, childname, childlistname)
  namep = data.GetAttributebyId(parent_id, "name")
  infotext = "You are going to create/add a component of type _
[" + childname + "] to [" + namep + "]. Continue with _
operation?"
  retval = Dialog.MessageBoxExt("Skript Information", _
infotext, "YES_NO")
  if retval <> 6 then
    REM Stopping operation
    sa_newchild=False
    call Dialog.MessageBox("Skript Information", "Creating _
new [" + childname + "] terminated.")
  else
    REM Continue operation
    sa_newchild=True
  end if
end function
```

### 4.1.2 ScriptActionAddChild

#### Description

The associated script is executed whenever a new child has been added to a parent by selecting **New** in the context menu. The parent\_id, the child\_id and the childlistname are passed to the script.

**Script Entry Function**

```
function sa_addchild(parent_id, child_id, childlistname)
```

Parameters	Description
parent_id	A string containing the ID of the parent object.
child_id	A string containing the ID of the child object which has been added (this is the ID of the BOM entry).
childlistname	<ul style="list-style-type: none"> <li>•Up to PE 5.13: A string containing either the name of the plantype, or "nodes", or the name of the type which has been created.</li> <li>•PE 5.14: A string containing either the GUID of the plantype or the name of the type which has been created.</li> <li>•PE 5.15 and later: For PPR components "nodes", for other types(relationships) the configured childlistname.</li> </ul>

**Script Action Example****Example**

```
function sa_addchild(parent_id, child_id, childlistname)
    namep = data.GetAttributeById(parent_id, "name")
    child_base_id = Data.GetAttributeById(child_id, "relationobject2")
    namec = data.GetAttributeById(child_id, "name")
    infotext = "You have added [" + namec + "] to [" + namep + "]"
    call Dialog.MessageBox("Script Information", infotext)
end function
```

**4.1.3 ScriptActionSetAttribute****Description**

The associated script is executed whenever the user has changed attributes in the **Properties** dialog. The object\_id and a list of all changed attribute-value pairs are passed to the script. The script can terminate the (before) event by returning 0, False, vbFalse (VBScript), or false(JScript). If this happens, the **Properties** dialog remains open. The old attribute values are not restored though.

**Script Entry Function**

```
function sa_setattribute(object_id, attributes)
```

Parameters	Description
object_id	A string containing the ID of the object whose attributes are going to be changed (have changed).
attributes	An array containing all attribute-value pairs the user has changed in the properties dialog.

**Script Action Example****Example**

```
function sa_setattribute(id, values)
    dimx = ubound(values,1)
    for i = 0 to dimx
        attribute = values(i,0)
        value = values(i,1)
        avlist = avlist + "[" + attribute + "] -->" + cstr(value) + vbCRLF
    next
    text = "You have changed the following attributes: " + vbCRLF + avlist + vbCRLF + "Continue?"
    retval = Dialog.MessageBoxExt("Script Warning", text, "YES_NO")
    if retval = 6 then
        sa_setattribute = True
    end if
end function
```

```

else
    sa_setattribute = False
end if
end function

```

The content of the attributes array is different for SA\_BEFORE and SA\_AFTER.

- In case of SA\_AFTER the values (not the attribute names) are completely set to zero (if setting all attributes is successful). This is a feature(!), because this zero means an operating system success code (equals S\_OK).

**Background:** If only one attribute in the array is incorrect or cannot be written, then the returned value for this attribute contains additional information about the error. This information could potentially be handled by the script.

#### 4.1.4 ScriptActionNew

##### Description

The associated script is executed whenever the user has created a new component.

##### Script Entry Function

**Alternative A** (all versions): `function sa_new(object_id)`

**Alternative B** (PE 5.14 SP2 and later): `function sa_new(object_id, link_id)`

Parameters	Description
object_id	A string containing the ID of the new object.
link_id	A string containing the ID of the new link object. This is the ID of a BOM entry if the new object is a PPR component; otherwise it is identical to object_id.

##### Script Action Example

##### Example

```

function sa_new(new_object_id)
    name = data.GetAttributeById(new_object_id, "name")
    call Dialog.MessageBox("The component [" + name + "] has been created.")
end function

```

#### 4.1.5 ScriptActionCopy

##### Description

The associated script is executed whenever the user copies one or more components. Passed to the script are the object ID of the parent, the object IDs of the components that should be (before) or have been (after) copied, and also the name of the childlist. The script can terminate the (before) event by returning 0, False, vbFalse (VBScript), or false(JScript).

##### Script Entry Function

`function sa_copy (parent_id, children_ids, childlistname, children_link_ids)`

Parameters	Description	
	Before	After
parent_id	A string containing the ID of the parent object.	

Parameters	Description	
	Before	After
children_ids	An array of strings containing the IDs of the (base objects) components to copy.	An array of strings containing the IDs of the copied components (link objects).
childlistname	A string containing the name of the childlist to which the copied children to (have been) append(ed).  •Up to PE5.13SP6/ PE5.14SP2: A string containing either the name of the plantype, or "nodes", or the name of the type(relationship) which has been created.  •PE5.13SP7/ PE5.14SP3 and later: A string containing either the GUID of the plantype or the name of the type(relationship) which has been created.	
children_link_ids	An array of strings containing the IDs of the (link objects) components to copy.	

### Script Action Example

#### Example

```
function sa_copy(parent_id, children_ids, childlistname)
  parentname=data.GetAttributebyId(parent_id, "name")
  dimx = ubound(children_ids,1)
  for i = 0 to dimx
    child_id = children_ids(i)
    name=data.GetAttributebyId(child_id, "name")
    copylist = copylist + name + vbCRLF
  next
  text = "You are going to copy the following components of parent [" + parentname + "]: " + vbCRLF + copylist + vbCRLF + "Continue?"
  retval = Dialog.MessageBoxExt("Script Warning", text, "YES_NO")
  if retval = 6 then
    sa_copy= True
  else
    sa_copy = False
  end if
end function
```

### Alternative Function Definition

```
function sa_copy(parent_id, children_ids, childlistname, children_link_ids) end function
```

## 4.1.6 ScriptActionLink

### Description

The associated script is executed whenever the user links one or more components to another parent object. Passed to the script are the object ID of the parent, the object IDs of the components that should be (before) or have been (after) linked, and also the name of the childlist. The script can terminate the (before) event by returning 0, False, vbFalse (VBScript), or false(JScript).

### Script Entry Function

```
function sa_link(parent_id, children_ids, childlistname, children_link_ids)
```

Parameters	Description	
	SA_BEFORE	SA_AFTER
children_ids	An array of strings containing the IDs of the (base objects) components to link.	An array of strings containing the IDs of the linked components (link objects).

Parameters	Description	
	SA_BEFORE	SA_AFTER
childlistname	<ul style="list-style-type: none"> <li>Up to PE5.13SP6/ PE5.14SP2: A string containing either the name of the plantype, or "nodes", or the name of the type(relationship) which has been created.</li> <li>PE5.13SP7/ PE5.14SP3 and later: A string containing either the GUID of the plantype or the name of the type(relationship) which has been created.</li> </ul>	
children_link_ids	An array of strings containing the IDs of the (link objects) components to link.	

### Script Action Example

#### Example

```
function sa_link(parent_id, children_ids, childlistname)
    parentname=data.GetAttributeById(parent_id, "name")
    dimx = ubound(children_ids,1)
    for i = 0 to dimx
        child_id = children_ids(i)
        name=data.GetAttributeById(child_id, "name")
        linklist = linklist + name + vbCRLF
    next
    text = "You are going to link the following components of parent [" + parentname + "]: " + vbCRLF + linklist + vbCRLF + "Continue?"
    retval = Dialog.MessageBoxExt("Script Warning", text, "YES_NO")
    if retval = 6 then
        sa_link= True
    else
        sa_link = False
    end if
end function
```

### Alternative Function Definition

```
function sa_link(parent_id, children_ids, childlistname, children_link_ids) end function
```

## 4.1.7 ScriptActionMove

### Description

The associated script is executed whenever the user moves one or more components from one parent object to another. Passed to the script are the object IDs of the old and new parent, the object IDs of the components that should be (before) or have been (after) moved, and also the name of the childlist. The script can terminate the (before) event by returning 0, False, vbFalse (VBScript), or false(JScript).

### Script Entry Function

```
function sa_move(oldparent_id, newparent_id, children_ids, childlistname)
```

Parameters	Description	
	SA_BEFORE	SA_AFTER
oldparent_id	A string containing the ID of the old parent object.	A string containing the ID of the old (up to PE5.13) new (PE5.14 and later) parent object.
newparent_id	A string containing the ID of the new parent object.	
children_ids	An array of strings containing the IDs	An array of strings containing the IDs

Parameters	Description	
	SA_BEFORE	SA_AFTER
	of the (link objects) components to link.	of the linked components (link objects).
childlistname	A string containing the name of the childlist to which the new children to (have been) append(ed).	

### Script Action Example

#### Example

```
function sa_move(oldparent_id, newparent_id, children_ids, childlistname)
oldparent_name=data.GetAttributebyId(oldparent_id, "name")
newparent_name=data.GetAttributebyId(newparent_id, "name")
dimx = ubound(children_ids,1)
for i = 0 to dimx
    child_id = children_ids(i)
    name=data.GetAttributebyId(child_id, "name")
    movelist = movelist + name + vbCRLF
next
text = "You are going to move the following components from parent [" + oldparent_name + "] to parent [" + newparent_name + "]. " + vbCRLF + movelist + vbCRLF + "Continue?"

retval = Dialog.MessageBoxExt("Script Warning", text, "YES_NO")
if retval = 6 then
    sa_move= True
else
    sa_move = False
end if
end function
```

## 4.1.8 ScriptActionDelete

### Description

The associated script is executed whenever the user is going to delete a component.



### Note

*If executed on a subcompitem or a relationship object, this scriptaction is only called if you select to delete the referenced object from the project library, too. Otherwise the scriptaction [RemoveComponent](#) is called. The object\_id of the object to delete is passed to the script. The script can terminate the event by returning 0, False, vbFalse (VBScript), or false(JScript).*

### Script Entry Function

*function sa\_delete(object\_id)*

Parameters	Description
object_id	A string containing the ID of the object that is going to be deleted.

### Script Action Example

#### Example

```
function sa_delete(object_id)
name = Data.GetAttributebyId(object_id, "name")
text = "This your last chance. You are going to delete the object <" + name + ">. Continue?"
retval = Dialog.MessageBoxExt("Script Warning", text, "YES_NO")
if retval = 6 then
    sa_delete = True
else
    sa_delete = False
end if
end function
```



```

    sa_delete = False
end if
end function

```

### Note

For PE5.15SP3 and later, the action is triggered **BEFORE** and **AFTER** the component is deleted. However, in the **AFTER** case, the passed ID is empty by design (because the object does not exist anymore).

If you need specific information related to the object that has been deleted, you have to implement a macro for the **BEFORE** action and store the required information temporarily, e.g. by means of a file, in order to read that information in the script launched by the **AFTER** action.

## 4.1.9 ScriptActionRemoveChild

### Description

The associated script is executed whenever the user is going to delete a linked object (subcompitem, relationship). The parent\_id (object in the tree view), the child\_id (object to delete) and the childlistname are passed to the script. The script can terminate the event by returning 0, False, vbFalse (VBScript), or false(JScript).

### Script Entry Function

**Alternative A** *function sa\_removechild (parent\_id, child\_id, childlistname)*

**Alternative B** (PE5.14SP2) *function sa\_removechild (parent\_id, child\_id, childlistname, child\_usage\_id)*

Parameters	Description	
parent_id	A string containing the ID of the parent object.	
child_id	A string containing the object ID of the component to be deleted.	
childlistname	<ul style="list-style-type: none"> <li>•For BOM entries: "nodes".</li> <li>•Otherwise: Type name or relationship name.</li> </ul>	
	SA_BEFORE	SA_AFTER
child_usage_id (PE5.14SP2 and later)	A string containing the object ID of the reference to be deleted. For PPR components, this is either the ID of a BOM entry or a relationship. Otherwise child_usage_id is equal to child_id.	identical to child_id

### Script Action Example

#### Example

```

function sa_removechild(parent_id, child_id, childlistname)
    namep = data.GetAttributebyId(parent_id, "name")
    namec = data.GetAttributebyId(child_id, "name")
    text = "This your last chance. You are going to remove the ob-
ject <" + namec + "> from the childlist [" + childlistname + "]
of parent [" + namep + "] Continue?"
    retval = Dialog.MessageBoxExt("Script Warning", text, "YES_NO")
    if retval = 6 then
        sa_removechild = True
    else
        sa_removechild = False
    end if
end function

```



### 4.1.10 ScriptActionInitProperties

#### Description

The associated script is executed whenever a properties dialog is opened. It will be launched **after** initialization of the controls from the database, but actually **before** the dialog itself is opened. Thus it is possible to overwrite values of specific controls and to append, remove, or replace values in combo box or list boxes.

The object IDs of the components whose attributes are shown in the dialog are passed to the script, refer to parameter description for details.

#### Script Entry Function

*function sa\_initproperties(object\_ids)*

Parameters	Description
Property sheet is displayed for .	Order of passed object IDs
BOM entries	1. ID of the base object referenced by the BOM entry \ 2. ID of the BOM entry (usage data)
Relationship objects	1. ID of the first related component 2. ID of the second related component 3. ID of the relationship object (usage data)
Other objects	ID of the object

#### Example

#### Script Action Example

```
function sa_initproperties(object_ids)
    attribute_id = "classification"
    text = "Attribute: " & attribute_id & vbCRLF & vbCRLF
    allvalues = Dialog.PropGetValues(object_ids(0), attribute_id , 0)
    dimx = ubound(allvalues, 1)
    for i = 0 to dimx
        visible_value = allvalues(i,0)
        internal_value = allvalues(i,1)
        text = text & visible_value & "(" & internal_value & ")" & vbCRLF
    next
    caption = "Value list"
    call Dialog.MessageBox(caption, text)

    CurrentValue = Dialog.PropGetCurrentValue(object_ids(0), _
    "classification")

    caption = "Current value"
    text = "Attribute: " & attribute_id & vbCRLF & "Current value: " _
    & CurrentValue
    call Dialog.MessageBox(caption, text)
    call set_carbodyposition(object_ids(0), CurrentValue)
    call set_nameshort(object_ids(0))
end function

REM *****
sub set_nameshort(object_id)
    Dim Value (0,0)
    Value (0,0) = "My Process (Script)"
    use_external = 1
    call Dialog.PropSetValues(object_id, "nameshort", use_external, _
    Value)
end sub

REM *****
sub set_carbodyposition(object_id, classification)
    Dim Values
```

```

' set carbodyposition menu for core processes
if classification= "c3" then
  ReDim Values(2,1)

  Values(0,0) = "to core process 1 (Script)"
  Values(0,1) = "cp1"

  Values(1,0) = "to core process 2 (Script)"
  Values(1,1) = "cp2"

  Values(2,0) = "to core process 3 (Script)"
  Values(2,1) = "cp3"

  use_external = 1
  call Dialog.PropSetValues(object_id, "carbodyposition", _
    use_external, Values)
end if
' set carbodyposition menu for standard processes
if classification = "c1" then
  ReDim Values(1,1)

  Values(0,0) = "to standard process 1 (Script)"
  Values(0,1) = "sp1"

  Values(1,0) = "to standard process 2 (Script)"
  Values(1,1) = "sp2"

  use_external = 1

  call Dialog.PropSetValues(object_id,"carbodyposition", _
    use_external, Values)
end if
' set carbodyposition menu for key processes
if classification = "c2" Then
  ReDim Values(0,1)
  Values(0,0) = "to key process (Script)"
  Values(0,1) = "kp1"
  use_external = 1
  call Dialog.PropSetValues(object_id,"carbodyposition", _
    use_external, Values)
end if

end sub

```

Please refer to the XscriptItemDialog [PropSetValues](#), XscriptItemDialog [PropGetValues](#), and XscriptItemDialog [PropGetCurrentValue](#)

### 4.1.11 ScriptActionChangeProperties

#### Description

The associated script is executed whenever the value of a combo box, list box control, edit control, multi line control, check box control, radio button control, and date control in the properties dialog is changed by the user. Thus it is possible to overwrite values of specific controls and to append, remove, or replace values in other combo boxes or list boxes.

The object ID of the affected component and the attribute ID (=name) are passed to the script, refer to parameter description for details.

#### Script Entry Function

*function [sa\\_changeproperties\(object\\_id, attribute\\_id\)](#)*

Parameters	Description
object_id	The ID of the affected object.

Parameters	Description
attribute_id	The ,name' of the attribute that has been changed.

**Script Action Example****Example**

```

function sa_changeproperties(object_id, attribute_id)

'refer to links below for function implementations
If attribute_id = "classification" then
    CurrentValue = Dialog.PropGetCurrentValue(object_id,
        "classification")
    call set_carbodyposition(object_id, CurrentValue)
End If

End function

REM *****
Sub set_carbodyposition(object_id, classification)
    Dim Values
    If classification= "c3" then ' set carbodyposition menu for core
processes
        ReDim Values(2,1)
        Values(0,0) = "to core process 1 (Script)"
        Values(0,1) = "cp1"

        Values(1,0) = "to core process 2 (Script)"
        Values(1,1) = "cp2"

        Values(2,0) = "to core process 3 (Script)"
        Values(2,1) = "cp3"

        use_external = 1
        call Di-
alog.PropSetValues(object_id,"carbodyposition",use_external, Values)
    End if
    If classification = "c1" then ' set carbodyposition menu for stan-
dard processes
        ReDim Values(1,1)

        Values(0,0) = "to standard process 1 (Script)"
        Values(0,1) = "sp1"

        Values(1,0) = "to standard process 2 (Script)"
        Values(1,1) = "sp2"

        use_external = 1

        call Di-
alog.PropSetValues(object_id,"carbodyposition",use_external, Values)
    End if
    If classification = "c2" Then ' set carbodyposition menu for key
processes
        ReDim Values(0,1)
        Values(0,0) = "to key process (Script)"
        Values(0,1) = "kp1"
        use_external = 1
        call Dialog.PropSetValues(object_id,"carbodyposition",
use_external, Values)
    End if
End sub

```

**4.1.12 ScriptActionNewVersion****Description**

The associated script is executed whenever the user is going to create (before) or has created (after) a new version of a component. The action is implicitly triggered on check out, too. The object ID is passed to the script. The script can terminate the event by returning 0, False, vbFalse (VBScript), or false(JScript).

#### Script Entry Function

*function sa\_newversion(object\_id)*

Parameter	Description
object_id	A string containing the ID of the object of which a new version is to be created (has been created).

#### Script Action Example

#### Example

```
function sa_newversion(object_id)
    object_name=Data.GetAttributebyId(object_id, "name")
    ps_id=Version.GetPlanningState(object_id)
    ps_name=Data.GetAttributebyId(ps_id, "name")
    text = "You are going to create a new version of object [" +
    object_name + "] currently having planning state [" + ps_name +
    "]. Continue?"
    retval = Dialog.MessageBoxExt("Script Warning", text,
    "YES_NO")
    if retval = 6 then
        sa_newversion= True
    else
        sa_newversion = False
    end if
end function
```

### 4.1.13 ScriptActionSetPlanningState

#### Description

The associated script is executed whenever the user is going to change (before) or has changed (after) the planningstate version of a component. The action is implicitly triggered on check in, too. The object ID and the ID of the new planning state are passed to the script.

The script can terminate the event by returning 0, False, vbFalse (VBScript), or false(JScript).

#### Script Entry Function

*function sa\_setplanningstate(object\_id, planningstate\_id)*

Parameters	Description
object_id	A string containing the ID of the object of which the planning state is to be changed (has been changed).
planningstate_id	A string containing the ID of the planning state to set.

#### Script Action Example

#### Example

```
function sa_setplanningstate(object_id, planningstate_id)
    object_name=Data.GetAttributebyId(object_id, "name")
    ps_name=Data.GetAttributebyId(planningstate_id, "name")
    text = You are going to set the planning state of object [" +
    object_name + "] to [" + ps_name + "]. Continue?"
    retval = Dialog.MessageBoxExt("Script Warning", text,
    "YES_NO")
    if retval = 6 then
        sa_setplanningstate= True
    else
```

```

    sa_setplanningstate = False
end if
end function

```

#### 4.1.14 ScriptActionOpenProject

##### Description

The associated script is executed after a project has been opened. You must select type "ergoproject" for this script action.

##### Script Entry Function

*function sa\_openproject(object\_id)*

Parameter	Description
object_id	A string containing the ID of the project which has been opened.

##### Script Action Example

##### Example

```

function sa_openproject(object_id)
    name = Data.GetAttributebyId(object_id, "name")
    text = "You have opened the project <" + name + ">."
    call Dialog.MessageBox("Info", text)
end function

```

#### 4.1.15 ScriptActionCloseProject

##### Description

The associated script is executed before a project is to be closed. The script can terminate the event by returning 0, False, vbFalse (VBScript), or false(JScript). You must select type "ergoproject" for this script action.

##### Script Entry Function

*function sa\_closeproject(object\_id)*

Parameter	Description
object_id	A string containing the ID of the project going to be closed.

##### Script Action Example

##### Example

```

function sa_closeproject(object_id)
    name = Data.GetAttributebyId(object_id, "name")
    text = "You are going to close project <" + name + ">. Continue?"
    retval = Dialog.MessageBoxExt("Script Warning", text, "YES_NO")
    if retval = 6 then
        sa_closeproject = True
    else
        sa_closeproject = False
    end if
end function

```

#### 4.1.16 ScriptAction SelectProject

##### Description

The associated script is executed after a project has been selected



##### Note

*You must select type "ergoproject" for this script action.*

**Script Entry Function**

```
function sa_selectproject(object_id)
```

Parameter	Description
object_id	A string containing the ID of the project which has been selected

**4.1.17 ScriptActionStartBalancing****Description**

The associated script is executed whenever a balancing sheet is opened. The resource ID (parent object) and the process ID(s) are passed to the script.

**Script Entry Function**

```
function sa_bal_start(resource_id, process_ids)
```

Parameters	Description
resource_id	A string containing the ID of the resource ID (parent object).
process_ids	A string array containing the IDs of the initial processes to be balanced.

**Example****Script Action Example**

```
function sa_bal_start(resource_id, process_ids)
    name = Data.GetAttributebyId(resource_id, "name")
    MsgBox("Resource Name " & name )

    dimx = ubound(ids)
    for i = 0 to dimx
        name = Data.GetAttributebyId(process_ids(i), "name")
        MsgBox("Process Name " & name )
    Next
end function
```

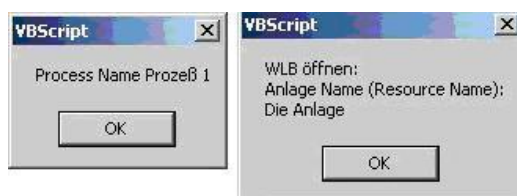


Figure 32: VB Script

**4.1.18 ScriptActionSaveBalancing****Description**

The associated script is executed whenever a balancing sheet is saved.

The resource ID (parent object), the deleted process ID(s), and the new process ID(s) are passed to the script.

**Script Entry Function**

```
function sa_bal_end (resource_id, deleted_process_ids, new_process_ids)
```

Parameters	Description
parent_id	A string containing the ID of the resource ID (parent object).
deleted_process_ids	A string array containing the IDs of the processes which have been deleted since the balancing sheet have been launched or saved.

Parameters	Description
new_process_ids	A string array containing the IDs of the processes which have been added since the balancing sheet have been launched or saved.

**Script Action Example****Example**

```
function sa_bal_end(resource_id, deleted_process_ids,
new_process_ids)
    name = Data.GetAttributeById(resource_id, "name")
    MsgBox("Resource Name " & name )

    dimx = ubound(deleted_process_ids)
    MsgBox("Arraysize Deleted: " & dimx)
    for i = 0 to dimx
        name = Data.GetAttributeById(deleted_process_ids(i), "name")
        MsgBox("Deleted Process Name " & name )
    Next

    dimx = ubound(new_process_ids)
    MsgBox("Arraysize New: " & dimx)
    for i = 0 to dimx
        name = Data.GetAttributeById(new_process_ids(i), "name")
        MsgBox("New Process Name " & name )
    Next
end function
```

**4.1.19 ScriptActionDropProcessToBalancing****Description**

The associated script is executed whenever a process or a list of processes is dropped into a balancing sheet.

**Script Entry Function**

*function sa\_bal\_drop\_proc(resource\_id, process\_ids)*

Parameters	Description
resource_id	A string containing the ID of the resource ID (parent object).
process_ids	A string array containing the IDs of the added processes.

**Script Action Example****Example**

```
function sa_bal_drop_proc(resource_id, process_ids)
    name = Data.GetAttributeById(resource_id, "name")
    MsgBox("Resource Name " & name )

    dimx = ubound(ids)
    for i = 0 to dimx
        name = Data.GetAttributeById(process_ids(i), "name")
        MsgBox("Process Name " & name )
    Next
end function
```

**4.1.20 ScriptActionPrint****Description**

The associated script is executed before and after running a print job. The object\_id of the object to print is passed to the script.

The script can terminate the event by returning 0, False, vbFalse (VBScript), or false(JScript).

**Script Entry Function**

*function sa\_print(object\_id)*

Parameter	Description
object_id	A string containing the ID of the object from which the print job is (has been) started.

**Script Action Example****Example**

```
function sa_print(object_id)
    name = Data.GetAttributeById(object_id, "name")
    text = "You are going to print from the object <" + name + ">. Con-
    tinue?"

    retval = Dialog.MessageBoxExt("Script Warning", text, "YES_NO")
    if retval = 6 then
        sa_print = True
    else
        sa_print = False
    end if
end function
```

**4.1.21 ScriptActionPrintlist****Description**

The associated script is executed before and after running a print job. The object\_id of the object to print is passed to the script. The script can terminate the event by returning 0, False, vbFalse (VBScript), or false(JScript).

**Script Entry Function**

*function sa\_printlist (object\_ids)*

Parameter	Description
object_id	A string containing the ID of the object from which the print job is (has been) started.

**Script Action Example****Example**

```
function sa_printlist(ids)
    dimx = ubound(ids, 1) + 1
    retval = MsgBox("SA PRINTLIST:" & dimx & " components selected!" &
    vbCRLF & "Display names?", 36)
    if retval = 6 then
        i=0
        Do while i < dimx
            value = Data.GetAttributeById (ids(i), "name")
            MsgBox(value)
            i = i+1
        Loop
    end if
end function
```

**4.1.22 ScriptActionOpenGraphic****Description**

The associated script is executed whenever the user is going to show or edit a graphics object.

The object\_id of the object to be displayed or edited is passed to the script. The script can terminate the event by returning 0, False, vbFalse (VBScript), or false(JScript).

**Script Entry Function**

*function sa\_opengraphic(object\_id)*

**Parameter**



Parameter	Description
object_id	A string containing the ID of the object.

### Script Action Example

#### Example

```
function sa_opengraphic(object_id)
name = Data.GetAttributebyId(object_id, "name")
text = "You are going to display the graphics of <" + name + ">.
Continue?"

retval = Dialog.MessageBoxExt("Script Warning", text, "YES_NO")
if retval = 6 then
    sa_opengraphic = True
else
    sa_opengraphic = False
end if

end function
```

## 4.1.23 Script Action RemoveChildEx

### Description

sa\_removechild is now extended to sa\_removechildex for supporting "simplerelation" as well.

The associated script is executed whenever you are going to delete a linked object (subcompitem, relationship, simplerelation). The parent\_id (object in the tree view), the child\_id (object to delete), and the childlistname are passed to the script. The script can terminate the event by returning 0, False, vbFalse (VBScript), or false(JScript).

### Script Entry Function


#### Alternative A


```
function sa_removechildex (parent_id, child_id,
childlistname)
```

#### Alternative B (PE5.14SP2)

```
function sa_removechildex (parent_id,
child_id, childlistname, child_usage_id)
```

### Parameter

Parameter	Description
object_id	A string containing the ID of the object.
parent_id	A string containing the ID of the parent object.
child_id	A string containing the object ID of the component to be deleted.
childlistname	<ul style="list-style-type: none"> <li>For BOM entries: "nodes".</li> <li>Otherwise: Type name or relationship name.</li> </ul>
child_usage_id (PE5.14SP2 and later)	<b>SA_BEFORE</b>
	<p>A string containing the object ID of the reference to be deleted.</p> <p>For PPR components, this is either the ID of a BOM entry or a relationship. Otherwise child_usage_id is equal to child_id.</p>
	<b>SA_AFTER</b>
	<p>identical to child_id</p> <p> <b>Note</b></p> <p>child_usage_id is NULL in case of simplerelation.</p>

Parameter	Description
	 <b>Note</b> <i>child_usage_id is NULL in case of simplereleation.</i>

**Example****Example**

```
function sa_removechildx(parent_id, child_id, childlistname)
namep = data.GetAttributebyId(parent_id, "name")
namec = data.GetAttributebyId(child_id, "name")
text = "This your last chance. You are going to remove the ob-
ject
<" + namec + "> from the childlist [" + childlistname + "]
of parent [" + namep + "] Continue?"
retval = Dialog.MessageBoxExt("Script Warning", text, "YES_NO")
if retval = 6 then
sa_removechildx = True
else
sa_removechildx = False
end if
end function
```

**4.1.24 Script Action Select Tree Item****Description**

After an object is selected (or selection changed) in PPR Navigator Tree a script action (sa\_selecttreeitem) would be executed.

**Script Entry Function**

*function sa\_selecttreeitem(object\_id)*

**Parameter**

Parameter	Description
object_id	A string containing the ID of selected object

**Example****Script Action Example**

```
function sa_selecttreeitem(object_id)

msgbox "Hello World!!!"

end function
```

**Note**

*The execution of "Select Tree Item" script action is not supported for "Before" event. When an object is selected or selection changed in PPR Navigator Tree, creation of list view and browser takes place. In case script action before event fails or script returns false. The creation of list view and browser would not takes place.*

**4.1.25 Script Action Expand Tree Item****Description**

After an object is expanded in PPR Navigator Tree a script action (sa\_expandtreeitem) would be executed.

**Script Entry Function**

```
function sa_expnadtreeitem(object_id)
```

**Parameter**

Parameter	Description
object_id	A string containing the ID of expanded object

**Example****Script Action Example**

```
function sa_expandtreeitem(id)

name = data.getattributebyid (id, "name")
msgbox "expanded tree item is----" & name

end function
```

**Note**

The execution of “Expand Tree Item” script action is not supported for “Before” event. When an object is expanded in PPR Navigator Tree, loading of children takes place. In case script action before event fails or script returns false. The loading of child components would not takes place.

## 5. Class Documentation XScriptingHost

The [Table 9](#) demonstrates the class documentation XScriptingHost.

**Table 9: Class Documentation**

Class	Named Item	Description
<a href="#">Class ScriptItemData</a>	Data	Managing data of PPR Hub objects. Get and set attributes, traverse through children lists, add, move, copy, and delete components.
<a href="#">Class ScriptItemQuery</a>	Query	Query the PPR hub.
<a href="#">Class ScriptItemVersion</a>	Version	Handling of object versions. Create, checkout, and use versions. (Re-)set planning states.
<a href="#">Class ScriptItemList</a>	List	Display a browser list view which acts as drag and drop source in the PPR Navigator.
<a href="#">Class XScriptItemDialog</a>	Dialog	User interaction via dialogs - message boxes, input box, and file selection.
<a href="#">Class ScriptItemGraphic</a>	Graphic	Retrieve the bounding box of a graphic object and create bitmaps (snapshots) of 3D objects.
<a href="#">Class ScriptItemGrid</a>	Sheet	Display a data sheet (grid).
<a href="#">Class SkriptItemRights</a>	Rights	Retrieve user information.
<a href="#">Delete Version Forced</a>	Unit	Retrieve unit information.
<a href="#">Class ScriptItemConvert</a>	Convert	Common conversion functions.
<a href="#">Class ScriptItemConfig</a>	Config	Retrieve information from configuration database.
<a href="#">Class ScriptItemLock</a>	CompLock	Lock and unlock components and get info concerning current locking state.

## 5.1 Class ScriptItemData

The [Table 10](#) demonstrates the class scriptitem data methods.

**Table 10: Class Script Item Data Methods**

Method	Description
<a href="#">GetAttributebyId</a>	Get a single attribute of an object.
<a href="#">SetAttributebyId</a>	Set a single attribute of an object.
<a href="#">GetAttributesbyId</a>	Get multiple attributes of an object by a single server access.
<a href="#">SetAttributesbyId</a>	Set multiple attributes of an object by a single server access.
<a href="#">GetLinkedObjectAttributebyId</a>	Get a single attribute of the object which is linked to this component.
<a href="#">SetLinkedObjectAttributebyId</a>	Set a single attribute of the object which is linked to this component.
<a href="#">GetLinkedObjectAttributesbyId</a>	Get multiple attributes of the object which is linked to this component by a single server access.
<a href="#">SetLinkedObjectAttributesbyId</a>	Set multiple attributes of the object which is linked to this component by a single server access.
<a href="#">GetFirstChild</a>	Retrieve a childlist of a component and return the first child in the list.
<a href="#">GetNextChild</a>	Retrieve the next child of a childlist (after a call to <a href="#">GetFirstChild</a> ).
<a href="#">GetChildrenCount</a>	Return the number of children in a list.
<a href="#">ResetIterator</a>	Reset the iterator of a distinct childlist back to the begin of the list (might be necessary if you use the same childlist more than once in a single script).
<a href="#">SetOrderAttribute</a>	Order the children of a list by a predefined attribute.
<a href="#">SetAttributeRange</a>	Define a range for a distinct attribute and retrieve all objects within that range.
<a href="#">DisableChildrenListFilter</a>	Locally disable all project filters in order to get the entire list of children.
<a href="#">CreateComponent</a>	Create a new component, and optionally a BOM entry and link this entry to the children list of the parent object.
<a href="#">AddComponent</a>	Create a BOM entry (=add an existing component to a children list) or create a relationship.
<a href="#">CopyComponent</a>	Copy a component.
<a href="#">RemoveComponent</a>	Remove a component from a childlist (use <b>ONLY</b> in combination with an <a href="#">AddComponent</a> statement to move a BOM entry. This function might become deprecated in future versions.)
<a href="#">DeleteComponent</a>	Delete a component (and implicitly remove it from (all) childlists).
<a href="#">CommitTransaction</a>	Commit the current transaction.
<a href="#">RollbackTransaction</a>	Rollback the current transaction.
<a href="#">CacheChildIds</a>	Define if child ids should be cached or not.
<a href="#">SetFetchingSize</a>	Select the number of children to fetch per server access.
<a href="#">IsDerivedFromClass</a>	Check if an object is derived from a distinct class (For example, "XDOErgoCompProcessDefault"). <i>Function is deprecated</i>
<a href="#">IsDerivedFromType</a>	Check if an object is derived from a distinct type (For example, "ergocompprocessdefault").
<a href="#">GetObjectGUIDbyObjectId</a>	Returns the objects Global Unique Identifier(GUID) given the

Method	Description
	objects ID.
<a href="#">GetObjectldbyObjectGUID</a>	Returns the objects ID given the Global Unique Identifier(GUID).
<a href="#">ExecuteServerMethod</a>	Execute a distinct server method (you have to pass an identifier).
<a href="#">CreateRelationshipWithOwner</a>	Create a so-called owner relationship (this replaces the former "connectionline" objects).
<a href="#">GetRelationshipsForOwnerByName</a>	Get the owner relationships of a distinct type (relation name).
<a href="#">GetOtherRelatedObject</a>	Get the second "parent" object of a relationship.
<a href="#">CopyAutoRelationUsageData</a>	Copies auto relation usage data from the source to the target object.
<a href="#">CreateRootOrLibraryComponent</a>	Create a project or a library component.
<a href="#">ReadSessionData</a>	Read session data from HKCU/HKLM or database(user dependent/global).
<a href="#">WriteSessionData</a>	Write session data to HKCU/HKLM or database(user dependent/global).
<a href="#">CopyTimeAnalysis</a>	Explicit copy of attached time analysis. (This function is not supported for PE5.17 and later any longer)
<a href="#">AttributeExists</a>	Check whether the passed attribute (name) exists for this object or not.
<a href="#">CreateRelationshipEx</a>	Creates a Relationship – between objects (exposed or unexposed) – with or without owner.
<a href="#">ChangeRelationship</a>	Changes participating objects of a relationship.
<a href="#">ChangeRelationshipEx</a>	Changes participating objects of a relationship according to the extended effectivity.
<a href="#">ExecuteServerMethodEx</a>	Execute a distinct server method (you have to pass an identifier).
<a href="#">GetRelationshipsForOwnerByNameEx</a>	Get the owner relationships of a distinct type (relation name) according to the extended effectivity.
<a href="#">OpenProperties</a>	Invokes the properties dialog of an object.
<a href="#">ResetItem</a>	Reset the script item to its initial state.

### 5.1.1 List of ScriptItemData Methods

#### 5.1.1.1 GetAttributebyId

##### Syntax

**GetAttributebyId (bstrObjectId, bstrName)**

Parameters	Description
bstrObjectId	A string containing the ID of the object.
bstrName	A string containing the name of the attribute to get.

##### Script Return Value

The current value of the attribute.

##### Example

#### Example

```
REM Get the current name of a component and
REM append the string "_VB" to name.
sub main (id)
    object_id=Data.GetAttributebyId(id, "ergocompbase")
    value = Data.GetAttributebyId(object_id, "name")
```

```

    value = value+"_VB"
    call Data.SetAttributebyId(object_id, "name", value)
end sub

```



### Note

*The get/set attribute(s) of subcompitem and relationship objects are most often redirected to an ergocompbase by server implementation.*

#### 5.1.1.2 SetAttributebyId

##### Syntax

**SetAttributebyId(bstrObjectId, bstrName, newValue)**

Parameters	Description
bstrObjectId	A string containing the ID of the object.
bstrName	A string containing the name of the attribute to set.
newValue	A variable of type variant containing the new value of the attribute.

##### Script Return Value

None

##### Example

### Example

```

REM Get the current name of a component and
REM append the string "_VB" to name.
sub main (id)
    object_id=Data.GetAttributebyId(id, "ergocompbase")
    value = Data.GetAttributebyId(object_id, "name")
    value = value+"_VB"
    call Data.SetAttributebyId(object_id, "name", value)
end sub

```



### Note

*Make sure that SetAttributebyId is always called on the "night" object. For instance, you cannot expect that the server automatically delegates the function call from a subcompitem to the ergocompbase in case the attribute is not implemented at the subcompitem.*

#### 5.1.1.3 GetAttributesbyId

##### Syntax

**GetAttributesbyId(bstrObjectId, pvarArrayOfNames)**

Parameters	Description
bstrObjectId	A string containing the ID of the object.
pvarArrayOfNames	An array containing the names of the attributes to get.

##### Example

```

REM Gets the attributes of object represented by "id" via a single database access
REM attributes : 1-dim array of attribute names
REM values: 2-dim array of results, dimy is 1 (attribute/value pairs)
REM all arrays are zero based, e.g. attributes(3) means 4 elements in the array
REM Note: the array values is sorted by attribute !
sub main(id)
    dim attributes (3)
    attributes(0)="name"
    attributes(1)="nameshort"
    attributes(2)="creationdate"
    attributes(3)="modificationdate"

```

```

values = Data.GetAttributesbyId(id, attributes)
dimx = ubound(values,1)
dimy = ubound( values, 2 ) ' must be 1 (!)
for i = 0 to dimx
    attribute = values( i, 0 )
    value = values( i, 1 )
    MsgBox(attribute & ": " & value)
next
end sub

```

#### 5.1.1.4 SetAttributesbyId

##### Syntax

**SetAttributesbyId(bstrObjectId, pVarArrayOfAttrValuePairs)**

Parameters	Description
bstrObjectId	A string containing the ID of the object.
pVarArrayOfAttrValuePairs	An array (or string) containing the attribute/value pairs to set.

##### Script Return Value

None

##### Example

#### Example

REM Sets the attributes of object represented by "id" via a single database access  
 REM attributes: 2-dim array of attribute/value pairs  
 REM arrays are zero based, e.g. attributes (1,1) means 2 elements in the 2-dim array

```

sub main(id)
    object_id=Data.GetAttributebyId(id, "ergocompbase")
    dim attributes (1, 1)
    attributes(0, 0) = "name"
    attributes(0, 1) = "Assembly"
    attributes(1, 0) = "nameshort"
    attributes(1, 1) = "123"
    call Data.SetAttributesbyId(object_id, attributes)
end sub

```



##### Note

*Make sure that SetAttributesbyId is always called on the "right" object. For instance, you cannot expect that the server automatically delegates the function call from a subcompitem to the ergocompbase in case the attribute is not implemented at the subcompitem.*

#### 5.1.1.5 GetLinkedObjectAttributebyId

##### Syntax

**GetLinkedObjectAttributebyId(bstrLinkObjectId, bstrParentObjectId, bstrName)**

Parameters	Description
bstrLinkObjectId	A string containing the ID of the relationship object.
bstrParentObjectId	A string containing the ID of one parent of the relationship object.
bstrName	A string containing the name of the attribute to get.

##### Script Return Value

The current value of the attribute.

##### Example

#### Example

REM Script illustrates the usage of-  
 Get/SetLinkedObjectAttribute(s) byId

```

REM Get the names of objects (type A) which are related to ob-
jects (type B) via relationname
REM Only one child is fetched to keep code more simple

sub main(parent_id)
    relationname="plant_provides_prod"
    parent_base_id=Data.GetAttributebyId(parent_id, "ergocompbase")
    child_id = Data.GetFirstChild(parent_base_id, relationname)
    if child_id <> "" Then
        name = Data.GetLinkedObjectAttributebyId(child_id, _
            parent_base_id, "name")
        name = name + " _VBS1"
        call Data.SetLinkedObjectAttributebyId(child_id, _
            parent_base_id, "name", name)
        MsgBox("Script executed successful." )
    else
        MsgBox("No relationship objects in list <" & relationname _
            & ">.")
    end if
end sub

```



### Note

*Use this method if you have an relationship object and one of its parents and you like to get an attribute of the other parent.*

#### 5.1.1.6 SetLinkedObjectAttributebyId

##### Syntax

**SetLinkedObjectAttributebyId(bstrLinkObjectId,  
bstrParentObjectId, bstrName, newValue)**

Parameters	Description
bstrLinkObjectId	A string containing the ID of the relationship object.
bstrParentObjectId	A string containing the ID of one parent of the relationship object.
bstrName	A string containing the name of the attribute to set.
newValue	A variable of type variant containing the new value of the attribute.

##### Script Return Value

None

### Example

#### Example

```

REM Script illustrates the usage of-
Get/SetLinkedObjectAttribute(s) byId
REM Get the names of objects (type A) which are related to ob-
jects (type B) via relationname
REM Only one child is fetched to keep code more simple
sub main(parent_id)
    relationname="plant_provides_prod"
    parent_base_id=Data.GetAttributebyId(parent_id, "ergocomp-
base")
    child_id = Data.GetFirstChild(parent_base_id, relationname)
    if child_id <> "" Then
        name = Data.GetLinkedObjectAttributebyId(child_id, par-
ent_base_id, "name")
        name = name + " _VBS1"
        call Data.SetLinkedObjectAttributebyId(child_id, par-
ent_base_id, "name", name)
        MsgBox("Script executed successful." )
    else
        MsgBox("No relationship objects in list <" & relationname &
">." )
    end if
end sub

```





```
end if
end sub
```

### Note

*Use this method if you have an relationship object and one of its parents and you like to set an attribute of the other parent.*

#### 5.1.1.7 GetLinkedObjectAttributesbyId

##### Syntax

**GetLinkedObjectAttributesbyId(bstrLinkObjectId,  
bstrParentObjectId, pvarArrayOfNames)**

Parameters	Description
bstrLinkObjectId	A string containing the ID of the relationship object.
bstrParentObjectId	A string containing the ID of one parent of the relationship object.
pvarArrayOfNames	An array containing the names of the attributes to get.

##### Script Return Value

An array of the attribute/value pairs to retrieve.

### Example

#### Example

```
REM Script illustrates the usage of Get / SetLinkedObjectAttribute(s)byId
REM Get the names of objects (type A) which are related to objects (type B) via relationname
REM Only one child is fetched to keep code more simple
sub main(parent_id)
    relationname="plant_provides_prod"
    parent_base_id=Data.GetAttributebyId(parent_id, "ergocomp-base")
    child_id = Data.GetFirstChild(parent_base_id, relationname)
    if child_id <> "" Then
        REM multiple attribute
        dim attributes (3)
        attributes(0)="name"
        attributes(1)="nameshort"
        attributes(2)="creationdate"
        attributes(3)="modificationdate"
        values = Data.GetLinkedObjectAttributesbyId(child_id, parent_base_id, attributes)
        dimx = ubound(values,1)
        for i = 0 to dimx
            attribute = values( i, 0 )
            value = values( i, 1 )
            MsgBox(Attribut & ": " & value)
        next
    else
        MsgBox("Kein relationship Objekt in der Liste <" & relationname >".")
    end if
end sub
```

#### 5.1.1.8 SetLinkedObjectAttributesbyId

##### Syntax

**SetLinkedObjectAttributesbyId(bstrLinkObjectId,  
bstrParentObjectId, pVarArrayOfAttrValuePairs)**

Parameters	Description
bstrLinkObjectId	A string containing the ID of the relationship object.

Parameters	Description
bstrParentObjectId	A string containing the ID of one parent of the relationship object.
pVarArrayOfAttrValuePairs	An array (or string) containing the attribute/value pairs to set.

**Script Return Value**

None

**Example****Example**

```

REM Script illustrates the usage of Get / SetLinkedObjectAttribute(s)byId
REM Get the names of objects (type A) which are related to objects (type B) via relationname
REM Only one child is fetched to keep code more simple
sub main(parent_id)
    relationname="plant_provides_prod"
    parent_base_id=Data.GetAttributebyId(parent_id, "ergocompbase")
    child_id = Data.GetFirstChild(parent_base_id, relationname)
    if child_id <> "" Then
        name = Data.GetLinkedObjectAttributebyId(child_id, parent_base_id, "name")
        nameshort = Data.GetLinkedObjectAttributebyId(child_id, parent_base_id, "nameshort")
        dim avpairs (1, 1)
        avpairs(0, 0) = "name"
        avpairs(0, 1) = name + "_VBS2"
        avpairs(1, 0) = "nameshort"
        avpairs(1, 1) = nameshort + "_VBS2"
        call Data.SetLinkedObjectAttributesbyId(child_id, parent_base_id, avpairs)
        MsgBox("Script executed successful." )
    else
        MsgBox("No relationship objects in list <" & relationname & ">." )
    end if
end sub

```

**5.1.1.9 GetFirstChild****Syntax****GetFirstChild(bstrParentObjectId, bstrChildListName)**

Parameters	Description
bstrParentObjectId	A string containing the object id of the parent.
bstrChildListName	A string containing the childlistname for which the children are to be fetched.

**Script Return Value**

The object id of the first child of the childlist or an empty string if there are no children to fetch.

**Example****Example**

```

REM Illustrates fetching children of childlist "nodes"
REM a) the children list is ordered (ascending) by name
REM b) before retrieving the children, the number of children is displayed
REM c) all children are fetched in the "Do while Loop"
REM d) after fetching all children the iterator is again set to the beginning of the list
sub main(parent_id)
    parent_id = Data.GetAttributebyId (parent_sci_id, "ergocomphase")
    childlistname = "nodes"
    call Data.SetOrderAttribute("name", "ASC")

```

```

count=Data.GetChildrenCount(parent_id, childlistname)
MsgBox("Number of children: " & count)
child_id = Data.GetFirstChild(parent_id, childlistname)
Do while child_id <> ""
    name = Data.GetAttributeById (child_id, "name")
    MsgBox("Name:" & name)
    child_id = Data.GetNextChild(parent_id, childlistname)
Loop
call Data.ResetIterator(parent_id, childlistname)
child_id = Data.GetNextChild(parent_id, childlistname)
nameshort = Data.GetAttributeById (child_id, "nameshort")
MsgBox("Short Name:" & nameshort)
end sub

```

For **R15SP3** and later, it is possible to parameterize the `GetFirstChild`, `GetNextChild`, and related commands so that not only one level of children (and relationships, respectively), but a list of children, grand children, etc. can be returned from the server in a single shot. However, in that case, the structure information is transferred from the server, i.e. it cannot be recognized on which level the child is positioned, since all children are simply listed without level information.

This is achieved by means of adding postfixes to the parameter **bstChildListName**.

- The postfix ***\_explicitandimplicit\_rs*** returns, similar to the regular call, the relationship (BOM entry) objects recursively.
- The postfix ***\_explicitandimplicit\_ro*** returns directly the base objects, i.e. there is no need to again iterate across the results and retrieve the base objects from the relationship (BOM entry) objects.

### Example

```

...
child_id = Data.GetFirstChild(parent_id,
"nodes_explicitandimplicit_rs")
Do while child_id <> ""
    [...]
    child_id = Data.GetNextChild(parent_id,
"nodes_explicitandimplicit_rs")
Loop
...

```

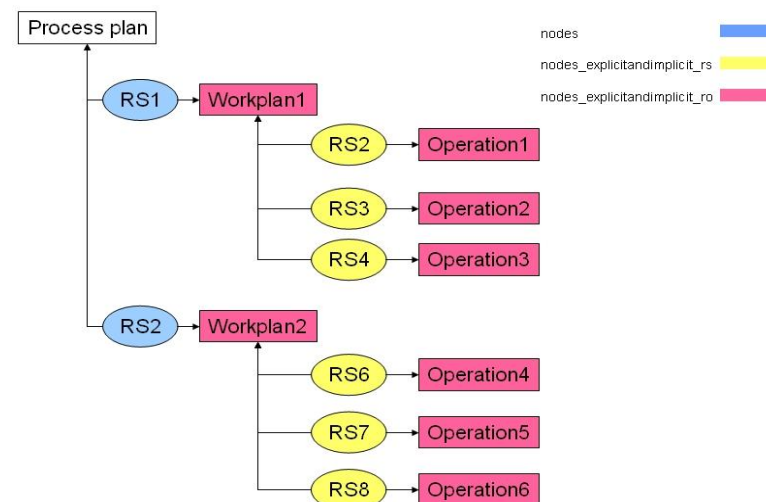


Figure 33: Get First Child

#### 5.1.1.10 GetNextChild

**Syntax****GetNextChild( bstrParentObjectId, bstrChildListName)**

Parameters	Description
bstrParentObjectId	A string containing the object id of the parent.
bstrChildListName	A string containing the childlistname for which the children are to be fetched.

**Script Return Value**

The object id of next child of the childlist or an empty string if there are no more children to fetch.

**Example****Example**

```

REM illustrates fetching children of childlist "nodes"
REM a) the children list is ordered (ascending) by name
REM b) before retrieving the children, the number of children is displayed
REM c) all children are fetched in the "Do while Loop"
REM d) after fetching all children the iterator is again set to the beginning of the list

sub main(parent_id)
    parent_id = Data.GetAttributebyId (parent_sci_id, "ergo comphase")
    childlistname = "nodes"
    call Data.SetOrderAttribute("name", "ASC")
    count=Data.GetChildrenCount(parent_id, childlistname)
    MsgBox("Number of children: " & count)
    child_id = Data.GetFirstChild(parent_id, childlistname)
    Do while child_id <> ""
        name = Data.GetAttributebyId (child_id, "name")
        MsgBox("Name:" & name)
        child_id = Data.GetNextChild(parent_id, childlistname)
    Loop
    call Data.ResetIterator(parent_id, childlistname)
    child_id = Data.GetNextChild(parent_id, childlistname)
    nameshort = Data.GetAttributebyId (child_id, "nameshort")
    MsgBox("Short Name:" & nameshort)
end sub

```

**5.1.1.11 GetChildrenCount****Syntax****GetChildrenCount (bstrParentObjectId, bstrChildListName)**

Parameters	Description
bstrParentObjectId	A string containing the object id of the parent.
bstrChildListName	A string containing the childlistname for which the children are to be fetched.

**Script Return Value**

The number of children within the childlist, or zero if there are no children in the list.

**Example****Example**

```

REM illustrates fetching children of childlist "nodes"
REM a) the children list is ordered (ascending) by name
REM b) before retrieving the children, the number of children is displayed
REM c) all children are fetched in the "Do while Loop"
REM d) after fetching all children the iterator is again set to the beginning of the list

sub main(parent_id)
    parent_id = Data.GetAttributebyId (parent_sci_id, "ergo comphase")

```

```

childlistname = "nodes"
call Data.SetOrderAttribute("name", "ASC")

count=Data.GetChildrenCount(parent_id, childlistname)
MsgBox("Number of children: " & count)
child_id = Data.GetFirstChild(parent_id, childlistname)
Do while child_id <> ""
    name = Data.GetAttributeById (child_id, "name")
    MsgBox("Name:" & name)
    child_id = Data.GetNextChild(parent_id, childlistname)
Loop
call Data.ResetIterator(parent_id, childlistname)
child_id = Data.GetNextChild(parent_id, childlistname)
nameshort = Data.GetAttributeById (child_id, "nameshort")
MsgBox("Short Name:" & nameshort)
end sub

```

#### 5.1.1.12 ResetIterator

##### Syntax

**ResetIterator(bstrParentObjectId, bstrChildListName)**

Parameters	Description
bstrParentObjectId	A string containing the object id of the parent.
bstrChildListName	A string containing the childlistname for which the children are to be fetched.

##### Script Return Value

If successful, the the iterator of childlist - defined by parent object id and childlistname - is set to the beginning of the list.

##### Example

#### Example

```

REM Illustrates fetching children of childlist "nodes"
REM a) the children list is ordered (ascending) by name
REM b) before retrieving the children, the number of children is displayed
REM c) all children are fetched in the "Do while Loop"
REM d) after fetching all children the iterator is again set to the beginning of the list

sub main(parent_id)
    childlistname = "nodes"
    call Data.SetOrderAttribute("name", "ASC")
    count=Data.GetChildrenCount(parent_id, childlistname)
    MsgBox("Number of children: " & count)
    child_id = Data.GetFirstChild(parent_id, childlistname)
    Do while child_id <> ""
        name = Data.GetAttributeById (child_id, "name")
        MsgBox("Name:" & name)
        child_id = Data.GetNextChild(parent_id, childlistname)
    Loop
    call Data.ResetIterator(parent_id, childlistname)
    child_id = Data.GetNextChild(parent_id, childlistname)
    nameshort = Data.GetAttributeById (child_id, "nameshort")
    MsgBox("Short Name:" & nameshort)
end sub

```

#### 5.1.1.13 SetOrderAttribute

##### Syntax

**SetOrderAttribute(bstrAttributeName, bstrChildListName);**

Parameters	Description
bstrAttributeName	The attribute which is a key for sorting or an empty string if sorting is not required.

Parameters	Description
bstrChildListName	Either "ASC" (ascending) or "DESC" (descending).

**Script Return Value**

None

**Example****Example**

```

sub main(id)
parent_base_id = Data.GetAttributebyId(id, "ergocompbase")
parent_name = Data.GetAttributebyId(parent_base_id, "name")
default_filename="D:\temp\ChildrenOf_" & parent_name &
"_SortedByName.txt"
caption="Select or create file ..."
Dim filter(0,1)
filter(0,0) = "Text Files (*.txt)"
filter(0,1) = "*.txt"
filename = Dialog.FileSelector("SaveAs", caption, de-
fault_filename, filter)
if filename <> "" then
Set fso = CreateObject("Scripting.FileSystemObject")
Set ts = fso.CreateTextFile (filename, True)
call Data.SetOrderAttribute("name", "DESC")
node_id = Data.GetFirstChild(parent_base_id, "nodes")
Do while node_id <> ""
node_base_id = Data.GetAttributebyId(node_id, "ergocompbase")
name = Data.GetAttributebyId(node_base_id, "name")
name = Data.GetAttributebyId(node_base_id, "name")
ts.WriteLine(node_base_id & " --> " & name)
node_id = Data.GetNextChild(parent_base_id, "nodes")
Loop
MsgBox("Script finished successful")
ts.Close
else
MsgBox("No file selected.")
end if
end sub

```

**5.1.1.14 SetAttributeRange****Syntax****SetOrderAttribute(bstrAttributeName, vLB, vUB)**

Parameters	Description
bstrAttributeName	The attribute which for which the range is defined.
vLB	The lower bound. This value is <i>included</i> in the range.
vUB	The upper bound. This value is <i>excluded</i> in the range.

**Note**

*If range parameters are unit values, they have to be passed in database units. For instance, the database unit for category time is seconds. If you need to convert from or to other units, please refer to the methods and functions of the script item [Delet Version Forced](#).*

**Script Return Value**

None

**Example****Example**

```

sub main(planttype_id)
planttype_name = Data.GetAttributebyId(planttype_id, "name")
default_filename="D:\temp\AllInstancesOf" & planttype_name & ".txt"
caption="Select or create file ..."

```

```

Dim filter(0,1)
filter(0,0) = "Text Files (*.txt)"
filter(0,1) = "*.txt"

filename = Dialog.FileSelector("SaveAs", caption, default_filename,
filter)
if filename <> "" then
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set ts = fso.CreateTextFile (filename, True)

    attributeName = "time"
    lowerBound = 60 'boundary value included
    upperBound = 120 'boundary value excluded

    call Data.SetAttributeRange(attributeName, lowerBound, upperBound )

    object_id = Data.GetFirstChild(plantype_id, "")
    Do while object_id <> ""
        name = Data.GetAttributebyId(object_id, "name")
        value = Data.GetAttributebyId(object_id, attributeName)
        ts.WriteLine(object_id & " --> " & name & " --> " & cstr(value))
        object_id = Data.GetNextChild(plantype_id, "")
    Loop
    MsgBox("Script finished successful")
    ts.Close
else
    MsgBox("No file selected.")
end if
end sub

```



### Note

The different modes for retrieving children list are mutually exclusive. The possible modes:

- Default (filtered according to the project settings, not sorted)
- Sorted (filtered according to the project settings, sorted), Please refer to the [SetOrderAttribute](#)
- Unfiltered (not sorted), Please refer to the [DisableChildrenListFilter](#)
- Range (unfiltered, unsorted, only objects having an attribute value within a predefined range are returned)

This method sets some variables which remain locally valid as long as the script is the running or one of the methods [SetOrderAttribute](#), [SetAttributeRange](#), [DisableChildrenListFilter](#), or [ResetItem](#) is called. Whenever a script is dealing with different children lists this has to be taken into account. The caller is responsible to set appropriate values. If the attribute provided is invalid for the children list under consideration, fetching children may fail, or the returned list may be empty.

Important: Using that mode, the functions [GetFirstChild](#) and [GetNextChild](#) are **limited** as follows:

The first parameter of the call is the object ID of a **slave** plantype(!) (the second parameter is ignored).

No project filter has been applied.

#### 5.1.1.15 DisableChildrenListFilter

##### Syntax

**SetOrderAttribute(bstrAttributeName, bFlag);**

Parameters	Description
bstrAttributeName	The attribute which for which the range is defined.
bFlag	<i>True</i> , if the children list should be returned unfiltered, otherwise <i>False</i> .



**Script Return Value**

None

**Example****Example**

```

sub main(id)

parent_base_id = Data.GetAttributebyId(id, "ergocompbase")
parent_name = Data.GetAttributebyId(parent_base_id, "name")
default_filename="D:\temp\ChildrenOf_" & parent_name &
"(Unfiltered).txt"
caption="Select or create file ..."

Dim filter(0,1)
filter(0,0) = "Text Files (*.txt)"
filter(0,1) = "*.txt"

filename = Dialog.FileSelector("SaveAs", caption, default_filename,
filter)
if filename <> "" then
Set fso = CreateObject("Scripting.FileSystemObject")
Set ts = fso.CreateTextFile (filename, True)

call Data.DisableChildrenListFilter(True)

node_id = Data.GetFirstChild(parent_base_id, "nodes")
Do while node_id <> ""
node_base_id = Data.GetAttributebyId(node_id, "ergocompbase")
name = Data.GetAttributebyId(node_base_id, "name")
name = Data.GetAttributebyId(node_base_id, "name")
ts.WriteLine(node_base_id & " --> " & name)
node_id = Data.GetNextChild(parent_base_id, "nodes")
Loop
MsgBox("Script finished successful")
ts.Close
else
MsgBox("No file selected.")
end if
end sub

```

**Note**

*The different modes for retrieving children list are mutually exclusive. The possible modes:*

- *Default (filtered according to the project settings, not sorted)*
- *Sorted (filtered according to the project settings, sorted), Please refer to the [SetOrderAttribute](#).*
- *Unfiltered (not sorted)*
- *Range (unfiltered, unsorted, only objects having an attribute value within a predefined range are returned), Please refer to the [SetAttributeRange](#).*

*This method sets some variables which remain locally valid as long as the script is the running or one of the methods [SetOrderAttribute](#), [SetAttributeRange](#), [DisableChildrenListFilter](#), or [ResetItem](#) is called. Whenever a script is dealing with different children lists this has to be taken into account. The caller is responsible to set appropriate values. If the attribute provided is invalid for the children list under consideration, fetching children may fail, or the returned list may be empty.*

**5.1.1.16 CreateComponent****Syntax**

**CreateComponent(bstrParentObjectId, bstrChildListName, bstrChildName)**

Parameters	Description
bstrParentObjectId	A string containing the ID of the parent object.



Parameters	Description
bstrChildListName	A string containing the name of the childlist to which the child should be appended.  If this parameter is "" (an empty string) the object is created in the project library
bstrChildName	A string containing the name of the type or plantype of the child to add.

**Script Return Value**

ID of the created object

**Example****Example (An Extract)**

```

sub main(id)
  ..
  plantype = "Part"
  childlistname = "nodes"
  grandparent_id=Data.GetAttributebyId(id, "ergocompbase")
  id_1 = Data.GetFirstChild(grandparent_id, childlistname)
  id_2 = Data.GetNextChild(grandparent_id, childlistname)
  parent_id_1=Data.GetAttributebyId(id_1, "ergocompbase")
  parent_id_2=Data.GetAttributebyId(id_2, "ergocompbase")
  ..
  'Create components directly as children of the parent 1
  sci_id = Data.CreateComponent(parent_id_1, childlistname, plan type)
  ....
end sub

```

**Note**

*Make sure that the plantype (or type) used in the script exists. Moreover, it must be possible to add this plantype (or type) as a child to the parent. You have to check your plantype (and/or type) configuration for appropriate parent child relations.*

**5.1.1.17 AddComponent****Syntax**

**AddComponent(bstrParentObjectId, bstrChildObjectId, BstrChildListName)**

Parameters	Description
bstrParentObjectId	A string containing the ID of the parent object or the first object to relate.
bstrChildObjectId	A string containing the ID of the child object to add or the second object to relate.
bstrChildListName	A string containing the name of the childlist to which the child should be appended or the relationship type.

**Script Return Value**

If successful, the ID of the created object. This is either the ID of:

- subcompitem, subcompviewitem (BOM handling)
- relationship (relationship handling).

**Example****Example for BOM handling (An Extract)**

```

sub main(id)
  ..
  plantype = "Part"
  childlistname = "nodes"
  grandparent_id=Data.GetAttributebyId(id, "ergocompbase")
  id_1 = Data.GetFirstChild(grandparent_id, childlistname)
  id_2 = Data.GetNextChild(grandparent_id, childlistname)
  parent_id_1=Data.GetAttributebyId(id_1, "ergocompbase")
  parent_id_2=Data.GetAttributebyId(id_2, "ergocompbase")
  ..
  'Create a component in the project library and add it to the childlist
  of parent 1
  comp_id = Data.CreateComponent(parent_id_1, "", plantype)

```

```
sci_id = Data.AddComponent(parent_id_1, comp_id, childlistname)
....
end sub
```

### 5.1.1.18 CopyComponent

#### Syntax

**CopyComponent(bstrParentObjectId, bstrChildCopyObjectId, iCopyMode)**

Parameters	Description
bstrParentObjectId	A string containing the ID of the parent object.
bstrChildCopyObjectId	A string containing the ID of the child object to copy.
iCopyMode	An integer that defines the copy mechanism.

Possible values for iCopyMode

Value	Description
0      Copy normal	Copy normal - copy the component only.
1      Copy flat	Copy flat - copy the component and its children(flat).
2      Copy deep	Copy deep - copy the component and its children(deep).

#### Script Return Value

ID of the created object.

#### Example (An Extract)

#### Example

```
sub main(id)
...
  plantype = "Part"
  childlistname = "nodes"
  grandparent_id=Data.GetAttributebyId(id, "ergocompbase")
  id_1 = Data.GetFirstChild(grandparent_id, childlistname)
  id_2 = Data.GetNextChild(grandparent_id, childlistname)
  parent_id_1=Data.GetAttributebyId(id_1, "ergocompbase")
  parent_id_2=Data.GetAttributebyId(id_2, "ergocompbase")
...
  'Make a copy of the last created component at parent 2
  sci_id = Data.CopyComponent(parent_id_2, comp_id, 0)
....
end sub
```

### 5.1.1.19 RemoveComponent

#### Syntax

**RemoveComponent(bstrParentObjectId, bstrChildObjectId, bstrChildListName)**

Parameters	Description
bstrParentObjectId	A string containing the ID of the parent object.
bstrChildObjectId	A string containing the ID of the child object to remove.
bstrChildListName	A string containing the name of the childlist from which the child is to remove.

#### Script Return Value

None

#### Example

#### Example (An Extract)

```
sub main(id)
...
  plantype = "Part"
  childlistname = "nodes"
  grandparent_id=Data.GetAttributebyId(id, "ergocompbase")
  id_1 = Data.GetFirstChild(grandparent_id, childlistname)
  id_2 = Data.GetNextChild(grandparent_id, childlistname)
  parent_id_1=Data.GetAttributebyId(id_1, "ergocompbase")
  parent_id_2=Data.GetAttributebyId(id_2, "ergocompbase")
...
end sub
```

```
'Move the last created component from parent 1 to parent 2
call Data.RemoveComponent(parent_id_1, sci_id, childlistname)
sci_id = Data.AddComponent(parent_id_2, sci_id, childlistname)
....
end sub
```



### Caution

Use this method only if you like to execute a move operation, i.e. remove a component from a parent and immediately add it to another parent.

Otherwise you create "dead" links in the database. If you like to remove a component permanently use [DeleteComponent](#).

#### 5.1.1.20 DeleteComponent

##### Syntax

**DeleteComponent(bstrParentObjectId, iDeleteMode)**

Parameters	Description
bstrParentObjectId	A string containing the ID of the parent object.
iDeleteMode	An integer defining the type of delete mechanism. (Please refer to the <a href="#">Caution</a> )

##### Script Return Value

None

##### Example (An Extract)

#### Example

```
sub main(id)
...
retval=Dialog.MessageBoxExt("Option for delete all created compo-
nents", "Delete from project library too?", "YES_NO_CANCEL")
if retval <> 2 then
if retval = 6 then
delete_flag=1
else
delete_flag=0
end if
Rem Delete the created components from the project tree (subcompitem)
and from the project library (ergocompbase) on demand
for i = 0 to 3
if delete_flag = 1 then
delete_id = GetBase(sci_ids(i))
Else
Delete_id = sci_ids(i)
End if
call Data.DeleteComponent(sci_ids(i), delete_flag)
next
end if
....
end sub
```



### Caution

Please take extreme care when calling this function recursively, in loops, and/or when setting the delete flag to "Deep".

Action that is performed dependend on object and iDeleteMode 0("flat") or 1("deep")

iDeleteMode	Object Base Configuration type	Result
0 (Flat)	subcompitem	Removes subcompitem only.
0 (Flat) or 1 (Deep)	relationship	Removes relationship only.
0 (Flat) or 1 (Deep)	ergoitem/dodefaultimpl	Removes ergoitem from project library.
0 (Flat)	ergocompbase	Removes ergocompbase from

iDeleteMode	Object Base Configuration type	Result
		project library and all referencing subcompitem.
1 (Deep)	ergocompbase or subcompitem	Removes ergocompbase and its children(!) from project library and all referencing subcompitem.

**Conclusion:** Usage of "Deep" only makes sense if you like to delete a parent including its children. Otherwise "Flat" is sufficient if you call "Delete" on the appropriate component.

### 5.1.1.21 CommitTransaction

#### Syntax

#### CommitTransaction(bstrObjectId)

Parameter	Description
bstrObjectId	A string containing any object ID within the current transaction.

#### Script Return Value

None

#### Example

#### Example

```
REM Example for commit and rollback of transactions
sub main (id)
    object_id=Data.GetAttributebyId(id, "ergocompbase")
    'Comit current transaction
    call Data.CommitTransaction(object_id)
    'Change an attribute, make a rollback, change the attribute again and
    commit the changes
    value = Data.GetAttributebyId(object_id, "name")
    value_false = value+"_JS"
    call Data.SetAttributebyId(object_id, "name", value_false)
    call Data.RollbackTransaction(object_id)
    value_true = value+"_VBS"
    call Data.SetAttributebyId(object_id, "name", value_true)
    call Data.CommitTransaction(object_id)
end sub
```



#### Note

*Please do not call this function within a script action call. This is because the function call may compete with the calling clients transaction concept. Misuse may cause corrupted database objects.*

### 5.1.1.22 RollbackTransaction

#### Syntax

#### RollbackTransaction(bstrObjectId);

Parameter	Description
BstrObjectId	A string containing any object ID within the current transaction.

#### Script Return Value

None

#### Example

#### Example

```
REM Example for commit and rollback of transactions
sub main (id)
    object_id=Data.GetAttributebyId(id, "ergocompbase")
    ' Commit current transaction
    call Data.CommitTransaction(object_id)
    ' Change an attribute, make a rollback, change the attribute again
    and commit the changes
    value = Data.GetAttributebyId(object_id, "name")
    value_false = value+"_JS"
    call Data.SetAttributebyId(object_id, "name", value_false)
```

```

call Data.RollbackTransaction(object_id)
value_true = value+"_VBS"
call Data.SetAttributebyId(object_id, "name", value_true)
call Data.CommitTransaction(object_id)
end sub

```

**Note**

*Please do not call this function within a script action call. This is because the function call may compete with the calling clients transaction concept. Misuse may cause corrupted database objects.*

**5.1.1.23 CacheChildIds****Syntax****CacheChildIds(bFlag)**

Parameters	Description
bFlag	A global flag which tells the host to cache the retrieved children in a map or not.

**Script Return Value**

None

**Example****Example (An Extract)**

```

sub main (object_id)
...
call Data.CacheChildIds(FALSE)
...
end sub

```

**Note**

*CacheChildIds(FALSE) is only suitable if you are traversing a single childlist at a time. It does not work in nested or recursive GetFirst/NextChild calls.*

**5.1.1.24 SetFetchingSize****Syntax****SetFetchingSize(ulFetchingSize);**

Parameters	Description
ulFetchingSize	The number of children to fetch in a single server access. Default is 25.

**Script Return Value**

None

**Example (An Extract)****Example**

```

REM Example for commit and rollback of transactions
sub main (object_id)
...
call Data.SetFetchingSize(50)
...
end sub

```

**5.1.1.25 IsDerivedFromClass****Syntax****IsDerivedFromClass(bstrObjectId, bstrClassString)**

Parameters	Description
bstrObjectId	A string containing the ID of the object.
bstrName	A string containing the class name to check for (you can pass the configured name as well as the real class name).

**Script Return Value**

**Example**

Parameters	Description
-1	If the object is derived from this class.
0	If the object is not derived from this class.

**Note**

*In VBScript -1 evaluates to True or vbTrue, but in JScript +1 evaluates to true.*

**Example**

```
sub main(id)
  name = Data.GetAttributebyId(id, "name")
  flag = ""
  retval = Data.IsDerivedFromClass(id, "XDOErgoProject")
  if retval = True then
    flag = "is a project."
  else
    flag = "is not a project."
  end if
  info = "The object <" + id + "> (" + name + ") " + flag
  MsgBox(info)
end sub
```

**5.1.1.26 IsDerivedFromType****Syntax**

**IsDerivedFromType(BstrObjectId, bstrTypeString)**

Parameters	Description
bstrObjectId	A string containing the ID of the object.
bstrTypeString	A string containing the configured type name to check for.

**Example**

```
sub main(project_id)
  name = Data.GetAttributebyId(id, "name")
  flag = ""
  retval = Data.IsDerivedFromType(id, "ergoproject")
  if retval = True then
    flag = "is a project."
  else
    flag = "is not a project."
  end if
  info = "The object <" + id + "> (" + name + ") " + flag
  MsgBox(info)
end sub
```

**5.1.1.27 GetObjectGUIDbyObjectId****SYNTAX**

**GetObjectGUIDbyObjectId(bstrObjectId)**

Parameter	Description
bstrObjectId	The objects ID.

**Script Return Value**

Parameter	Description
bstrObjectGUID	The objects Global Unique Identifier (GUID).

**Example****Example**

```
sub main(project_id)
  project_guid = Data.GetObjectGUIDbyObjectId(project_id)
  MsgBox("The projects GUID is: " & project_guid)
  project_id2 = Data.GetObjectIdbyObjectGUID(project_guid, "ergoproject")
  if project_id = project_id2 then
    MsgBox("Success. The returned ID is identical to the entry id.")
  else
    MsgBox("Failure.")
  end if
end sub
```

**Note**

To identify an object the software uses two different IDs, the (POET) object ID and a generated Global Unique Identifier. On upgrading or migrating a database the (POET) object ID can be subjected to a change, while the GUID remains constant.

**5.1.1.28 GetObjectldbyObjectGUID****SYNTAX**

**GetObjectldbyObjectGUID(bstrObjectGUID, bstrTypeName)**

Parameters	Description
bstrObjectGUID	The objects Global Unique Identifier (GUID).
bstrTypeName	The objects unique typename.

**Example****Example**

```
sub main(project_id)
    project_guid = Data.GetObjectGUIDbyObjectId(project_id)
    MsgBox("The projects GUID is: " & project_guid)
    project_id2 = Data.GetObjectIdbyObjectGUID(project_guid, "ergopro-
    ject")
    if project_id = project_id2 then
        MsgBox("Success. The returned id is identical to the entry id.")
    Else
        MsgBox("Failure.")
    end if
end sub
```

**Note**

To identify an object the software uses two different IDs, the (POET) object ID and a generated Global Unique Identifier. On upgrading or migrating a database the (POET) object ID can be subjected to a change, while the GUID remains constant

**5.1.1.29 ExecuteServerMethod****SYNTAX**

**GetObjectldbyObjectGUID(bstrObjectId, bstrMethod);**

Parameters	Description
bstrObjectId	The object ID for which the function must be executed.
bstrMethod	A string identifier describing the function (branch) to be executed.

**Example****Example**

```
sub main(id)
    call Data.ExecuteServerMethod(id, "freqrecalc")
end sub
```

**Note**

This call triggers a special implementation in the server which might be specific for the object type of the passed object (ID). The passed function identifier (i.e. "freqcalc") specifies the branch which have to be executed.

**CreateRelationshipWithOwner****SYNTAX**

**CreateRelationshipWithOwner(bstrRelationName, bstrSourceObjectId, bstrTargetObjectId, bstrOwnerObjectId)**

Parameters	Description
bstrRelationName	The name of the relationship to create (For example, "process_runsbefore_process").

Parameters	Description
bstrSourceObjectId	The source object ID.
bstrTargetObjectId	The target object ID.
bstrOwnerObjectId	The object ID of the base object that owns the graph.

**Script Return Value**

Parameter	Description
bstrRelationshipObjectId	The ID of the relationship object that has been created.

**Example****Example**

```
sub main(id)
...
relationship_id = Data.CreateRelationshipWithOwner(relationship_name,
child1_proc_id, child2_proc_id, parent_proc_id)
...
end sub
```

**Note**

*This function is required to create a relationship between processes in the processgraph. Actually such relationships are managed using "connectionlines" in the previous PE versions. The "connectionline" concept has been removed with PE 5.10.*

**5.1.1.30 GetRelationshipsForOwnerByName****SYNTAX**

**CreateRelationshipWithOwner(bstrSourceObjectId, bstrOwnerObjectId, bstrRelationName)**

Parameters	Description
bstrSourceObjectId	The ID of the child object for which relationships to other children are to be retrieved.
bstrOwnerObjectId	The object ID of the base object that owns the graph.
bstrRelationName	The name of the relationship e.g. "process_runsbefore_process".

**Script Return Value**

Parameter	Description
vRelationshipIds	An array of object IDs representing all relationships between a distinct child and the other children of the graph owning base object.

**Example****Example**

```
Dim relationship_name
sub main(id)
relationship_name = "process_runsbefore_process"
parent_process_id = GetBase(id)

child_process_sci_id = Data.GetFirstChild(parent_process_id, "nodes")
Do while child_process_sci_id <> ""
child_process_id = GetBase(child_process_sci_id)
child_process_name = GetName(child_process_id)

rel_ids = Data.GetRelationshipsForOwnerByName(child_process_id, parent_process_id, relationship_name)
dimx = ubound(rel_ids)
if dimx >= 0 then
for i=0 to dimx
other_process_id = Data.GetOtherRelatedObject(rel_ids(i), child_process_id)
other_process_name = GetName(other_process_id)
```



```

MsgBox("<" & child_process_name & "> has a relation of type <" & relationship_name & "> to <" & other_process_name & ">.")
next
else
MsgBox("<" & child_process_name & "> has no relations of type <" & relationship_name & "> to other subprocesses.")
end if
child_process_sci_id = Data.GetNextChild(parent_process_id, "nodes")
Loop
end sub
'*****
function GetBase(id)
GetBase = Data.GetAttributebyId(id, "ergocompbase")
end function
'*****
function GetName(id)
GetName = Data.GetAttributebyId(id, "name")
end function

```



### Note

*Beyond this special function other compilations of relationship objects can be retrieved by means of a query.*

#### 5.1.1.31 GetOtherRelatedObject

##### SYNTAX

**CreateRelationshipWithOwner (bstrRelationshipObjectId, bstrObjectOneld)**

Parameters	Description
bstrRelationshipObjectId	The object ID of the relationship object.
bstrObjectOneld	The object ID of the "first" object.

##### Script Return Value

Parameter	Description
bstrObjectTwold	The object ID of the "second" object ("the other object") which is associated via the relationship object.

##### Example

##### Example

```

sub main(id)
relationship_name = "process_runsbefore_process"
parent_process_id = GetBase(id)
child_process_sci_id = Data.GetFirstChild(parent_process_id, "nodes")
Do while child_process_sci_id <> ""
    child_process_id = GetBase(child_process_sci_id)
    child_process_name = GetName(child_process_id)
    rel_ids = Data.GetRelationshipsForOwnerByName(child_process_id, parent_process_id, relationship_name)
    dimx = ubound(rel_ids)
    if dimx >= 0 then
        for i=0 to dimx
            other_process_id = Data.GetOtherRelatedObject(rel_ids(i), child_process_id)
            other_process_name = GetName(other_process_id)
            MsgBox("<" & child_process_name & "> has a relation of type <" & relationship_name & "> to <" & other_process_name & ">.")
        next
    else
        MsgBox("<" & child_process_name & "> has no relations of type <" & relationship_name & "> to other subprocesses.")
    end if
    child_process_sci_id = Data.GetNextChild(parent_process_id, "nodes")
Loop
end sub
'*****
function GetBase(id)
GetBase = Data.GetAttributebyId(id, "ergocompbase")
end function
'*****
function GetName(id)
GetName = Data.GetAttributebyId(id, "name")
end function

```



### Note

*This function allows you to easily retrieve the ID of the second "the other" object of a relationship given the ID of first object is already known. It does basically the same as:*

```
secondobject_id = GetLinkedObjectAttributebyId(relationship_id, firstobject_id, "oid")
```

#### 5.1.1.32 CopyAutoRelationUsageData

##### SYNTAX

**CopyAutoRelationUsageData(bstrSourceObjectId, bstrTargetObjectId)**

Parameter	Description
bstrSourceObjectId	The ID of the source object.
bstrTargetObjectId	The ID of the target object.

##### Script Return Value

None

### Example

#### Example

```
sub main(id)
    rel_ids = Data.CopyAutoRelationUsageData(source_id, target_id)
end sub
```

#### 5.1.1.33 CreateRootOrLibraryComponent

##### SYNTAX

**CreateRootOrLibraryComponent(bstrRootObject, bstrParam)**

Parameters	Description
bstrRootObject	The identifier for the root object. Possible values are "ProjectRoot" or "ArchivRoot".
bstrParam	If bstrRootObject = "ProjectRoot" then you have to specify the ID of the (master) plantypeset for which the project should be created.

##### Script Return Value

The ID of the created object. This is either the ID of a project or library component.

#### Example

### Example

```
function CreateProject
```

```
Dim PTSCombo
Query.ResetSearch
Call Query.SetQuery("ergoplantypeset", "m_pMaster", "=", Empty)
Call Query.SetConcatenator("AND")
Call Query.SetQuery("ergoplantypeset", "tablename", "<>", "ergoplantypeset_wsc")

number_of_results = Query.GetResultCount
If number_of_results > 0 Then
    Redim PTSCombo(number_of_results, 1)
    i = 0

    result_id = Query.GetFirstResult
    Do While result_id <> ""
        Name = Data.GetAttributebyId(result_id, "name")
        PTSCombo(i, 0) = Name
        PTSCombo(i, 1) = result_id
        i = i + 1
        result_id = Query.GetNextResult
    Loop
    Call Dialog.CreateInputControl("ProjectName", "EditString",
    "New Project Name", "MyNewProject")
```

```

    Call Dialog.CreateInputControl("PTS", "ComboBox", "Plantypesets",
    PTSCombo)

    ret = Dialog.InputBox("Select a plantypeset")

    If ret <> 0 Then
        'check content of edit controls by means of control id (see
        above)
        master_pts_id = Dialog.GetInputControlValue("PTS")
        ProjectName = Dialog.GetInputControlValue("ProjectName")
        If master_pts_id <> "" Then
            project_id = Data.CreateRootOrLibraryComponent("ProjectRoot",
            master_pts_id)
            Call Data.SetAttributebyId(project_id, "name", ProjectName)
            CreateProject = project_id
            MsgBox ("Project created successfully.")
        Else
            MsgBox ("Invalid Master PTS ID.")
        End If

    Else
        MsgBox ("No PTS selected.")
    End If

Else
    MsgBox ("No PTS list to select from.")
End If
End Function

```



### Note

*This function allows a user to create projects and library components, for instance project independent scripts. Use this function with care - some objects have to be initialized properly to be visible in the user-interface. If you are not sure about this, do not try to implement such a function!.*

## 5.1.1.34 ReadSessionData

### Syntax

**ReadSessionData(bstrKey, bstrAttribute, bstrLocation)**

Parameters	Description
bstrKey	The subkey relative to "DELMIA\ergoplan" (registry) or "ergoplan" (database).
bstrAttribute	The name of the attribute.
bstrLocation	The name of the location (storage).

Possible values for **bstrLocation**

Value	Description
HKCU	Registry HKEY_CURRENT_USER
HKLM	Registry HKEY_LOCAL_MACHINE
SETTINGSDEFAULT	Registry Defaults HKEY_LOCAL_MACHINE option-configuration
DBUSER	Database PE User Dependent
DBGLOBAL	Database Global

### Script Return Value

The (string) value of the attribute for which key and location have been provided. If the provided key/attribute combination does not exist, the return value is empty.

### Example

#### Example

```

sub main(notused)
    subkey_default = "common"
    call Dialog.CreateInputControl("Subkey", "EditString", "Subkey",
    subkey_default)

```

```

attribute default = "languageid"
call Dialog.CreateInputControl("Attribute", "EditString", "Attribute",
attribute_default)
Dim combo(4,1)
combo(0,0) = "Registry HKEY_CURRENT_USER"
combo(0,1) = "HKCU"
combo(1,0) = "Registry HKEY_LOCAL_MACHINE"
combo(1,1) = "HKLM"
combo(2,0) = "Registry Defaults HKEY_LOCAL_MACHINE option-
configuration"
combo(2,1) = "SETTINGSDEFAULT"
combo(3,0) = "Database PE User Dependent"
combo(3,1) = "DBUSER"
combo(4,0) = "Database Global"
combo(4,1) = "DBGLOBAL"
call Dialog.CreateInputControl("Location", "ComboBox", "Location", com-
bo)
ret = Dialog.InputBox("Read Session Data")
if ret <> 0 then
'check content of edit controls by means of control id (see above)
subkey=Dialog.GetInputControlValue("Subkey")
attrib=Dialog.GetInputControlValue("Attribute")
location=Dialog.GetInputControlValue("Location")
value = Data.ReadSessionData(subkey, attrib, location)
...
end if
end sub

```

The Process Engineer user interface provides session data access through "Tools/Settings/Maintenance Tool...".

### 5.1.1.35 WriteSessionData

#### Syntax

**WriteSessionData(bstrKey, bstrAttribute, bstrLocation, vValue)**

Parameters	Description
bstrKey	The subkey relative to "DELMIA\ergoplan" (registry) or "ergoplan" (database).
bstrAttribute	The name of the attribute.
bstrLocation	The name of the location (storage).
vValue	The value to be stored.

#### Script Return Value

The (string) value of the attribute for which key and location have been provided. If the provided key/attribute combination does not exist, the return value is empty.

#### Possible values for bstrLocation

Values	Description
HKCU	Registry HKEY_CURRENT_USER
HKLM	Registry HKEY_LOCAL_MACHINE
SETTINGSDEFAULT	Registry Defaults HKEY_LOCAL_MACHINE option-configuration
DBUSER	Database PE User Dependent
DBGLOBAL	Database Global

#### Example

#### Example

```

sub main(notused)
...

call Dialog.CreateInputControl("Subkey", "EditString", "Subkey", sub-
key)

```

```

call Dialog.ModifyInputControl("Subkey", "ReadOnly", True)

call Dialog.CreateInputControl("Attribute", "EditString", "Attribute",
attrib)
call Dialog.ModifyInputControl("Attribute", "ReadOnly", True)

call Dialog.CreateInputControl("Location", "EditString", "Location",
location)
call Dialog.ModifyInputControl("Location", "ReadOnly", True)

call Dialog.CreateInputControl("NewValue", "EditString", "New Value",
value)
ret = Dialog.InputBox("Modify Session Data")
if ret <> 0 then
    new_value = Dialog.GetInputControlValue("NewValue")
    call Data.WriteSessionData(subkey, attrib, location, new_value)
    ...
end if

end sub

```

### 5.1.1.36 CopyTimeAnalysis

#### Syntax

**CopyTimeAnalysis (bstrSourceObjectId, bstrTargetObjectId)**

Parameters	Description
bstrSourceObjectId	A string containing the ID of the source object.
bstrTargetObjectId	A string containing the ID of the target object.

#### Script Return Value

None

#### Example (An Extract)

#### Example

```

sub main(notused)
...
target_process_sci_id = Data.CopyComponent(parent_process_id,
source_process_id, 0)
target_process_id = Data.GetAttributeById(target_process_sci_id, "erg-
combase")
call Data.CopyTimeAnalysis(source_process_id, target_process_id)<
...

```



#### Note

*This function is required to explicitly copy the attached time analysis (non-integrated standard time measurement client).*

*(This function is not supported for PE5.17 and later any longer)*

### 5.1.1.37 AttributeExists

#### Syntax

**AttributeExists(bstrObjectId, bstrAttributeName);**

Parameters	Description
bstrObjectId	A string containing the ID of the object.
bstrAttributeName	A string containing the class name to check for (you can pass the configured name as well as the real class name)

#### Script Return Value

Parameters	Description
-1	If the attribute does exist for this particular object.
0	If the attribute does not exist for this particular object.

#### Example (An Extract)

**Example**

```

sub main(notused)
    name = Data.GetattributebyId(id, "name")
    flag= ""
    retval = Data.AttributeExists(id, "myattr")
    if retval = True then
        flag = "does exist."
    Else
        Flag = "does not exist."
    End if
    Info = "The attribute myattr of object <" & id & "> (" & name & ") " &
flag
    MsgBox(info)
end sub

```

**5.1.1.38 CreateRelationshipEx****SYNTAX**

**CreateRelationshipEx(bstrRelationName,bstrSourceObjectId,bstrTargetObjectId,bstrSourceUnexposedObjectId, bstrTargetUnexposedObjectId, bstrOwnerObjectId)**

Parameters	Description
bstrRelationName	The name of the relationship to create (For example, "process_runsbefore_process")
bstrSourceObjectId	The source object ID.
bstrTargetObjectId	The target object ID.
bstrSourceUnexposedObjectId	The source unexposed object ID
bstrTargetUnexposedObjectId	The target unexposed object ID.
bstrOwnerObjectId	The object ID of the base object that owns the graph.

**Example****Example**

```

REM This method creates a subassembly and two operations in the project-
tlibrary
REM It creates a relationship between operation1 and Subassyl
REM Afterwards it replaces the participating object operation1 of the
relationship by Operation2
sub main (parent_id)
    ' call this script on projects only
    subassyl = CreateComp (parent_id, "", "Subassembly", "Subassyl")
    operation1 = CreateComp (parent_id, "", "Operation", "Operation1")
    operation2 = CreateComp (parent_id, "", "Operation", "Operation2")
    ' create relationship (Operation1, "") -----
    process_implements_requirement -----> (Subassyl,
    "unique_id_for_req_1")
    relship1 = data.CreateRelationshipEx("process_implements_requirement",
    operation1, subassyl, "", _ "unique_id_for_req_1", "")
    relship1_info = data.GetAttributeById(relship1, "relationshipinfo")
    MsgBox "relationship successfully created: " & vbNewLine & rel-
    ship1_info
    ' change given relationship to
    ' create relationship (Operation2, "") -----
    process_implements_requirement -----> (Subassyl, "changed_id")
    ' bitmask of cmSourceExposed (1) and cmTargetUnexposed (8) ==> 9
    call Data.ChangeRelationship(9, relship1, operation2, "", "",
    "changed_id", "")
    relship1_info = data.GetAttributeById(relship1, "relationshipinfo")
    MsgBox "relationship successfully changed to: " & vbNewLine & rel-
    ship1_info
end sub

function CreateComp (parent_id, childlistname, childname, namecomp)
    comp = data.CreateComponent(parent_id, childlistname, childname)
    data.SetAttributebyId comp, "name", namecomp
    CreateComp = comp

```

```
end function
```

### Script Return Value

Parameters	Description
bstrRelationshipObjectId	The ID of the relationship object that has been created.

## 5.1.1.39 ChangeRelationship

### SYNTAX

**ChangeRelationship(IngChangeMask,bstrRelationshipObjectId,bstrSourceObjectId,bstrTargetObjectId,  
bstrSourceUnexposedObjectId,bstrTargetUnexposedObjectId,  
bstrOwnerObjectId)**

Parameters	Description
IngChangeMask	Bitmask specifying the object to change.
bstrSourceObjectId	The source object ID.
bstrTargetObjectId	The target object ID.
bstrSourceUnexposedObjectId	The source unexposed object ID.
bstrTargetUnexposedObjectId	The target unexposed object ID.
bstrOwnerObjectId	The object ID of the base object that owns the graph.

### Example

```
REM This method creates a subassembly and two operations in the project-
tlibrary
REM It creates a relationship between operation1 and Subassyl
REM Afterwards it replaces the participating object operation1
of the relationship by Operation2
sub main (parent_id)
' call this script on projects only
subassyl = CreateComp (parent_id, "", "Subassembly", "Subassyl")
operation1 = CreateComp (parent_id, "", "Operation", "Operation1")
operation2 = CreateComp (parent_id, "", "Operation", "Operation2")
' create relationship (Operation1, "") ----
process_implements_requirement -----> (Subassyl,
"unique_id_for_req_1")
relship1 = data.CreateRelationshipEx("process_implements_requirement", operation1, subassyl, "", _
"unique_id_for_req_1", "")
relship1_info = data.GetAttributeById(relship1, "relationshipinfo")
MsgBox "relationship successfully created: " & vbNewLine & relship1_info
' change given relationship to
' create relationship (Operation2, "") ----
process_implements_requirement -----> (Subassyl, "changed_id")
' bitmask of cmSourceExposed (1) and cmTargetUnexposed (8) ==> 9
call Data.ChangeRelationship(9, relship1, operation2, "", "",
"changed_id", "")
relship1_info = data.GetAttributeById(relship1, "relationshipinfo")
MsgBox "relationship successfully changed to: " & vbNewLine & relship1_info
end sub

function CreateComp (parent_id, childlistname, childname, namecomp)
comp = data.CreateComponent(parent_id, childlistname, childname)
```

```
data.SetAttributebyId comp, "name", namecomp
CreateComp = comp
end function
```

**Script Return Value**

Parameter	Description
bstrRelationshipObjectId	The ID of the relationship object that has been created.

**Note**

*Changes participating objects of a relationship. This method should be used to change one or more objects participating in a relationship.*

**5.1.1.40 ChangeRelationshipEx****Syntax**

**ChangeRelationshipEx (lngChangeMask, bstrRelationshipObjectId, bstrSourceObjectId, bstrTargetObjectId, bstrSourceUnexposedObjectId, bstrTargetUnexposedObjectId, bstrOwnerObjectId, bstrExtEffectivity)**

Parameters	Description
lngChangeMask	Bitmask specifying the object to change.
bstrSourceObjectId	The source object ID.
bstrTargetObjectId	The target object ID.
bstrSourceUnexposedObjectId	The source unexposed object ID.
bstrTargetUnexposedObjectId	The target unexposed object ID.
bstrOwnerObjectId	The object ID of the base object that owns the graph.
bstrExtEffectivity	The given extended effectivity.

**Script Return Value**

Parameter	Description
bstrRelationshipObjectId	The ID of the relationship object that has been created

**Example****Example**

```
REM This method creates a subassembly and two operations in the
projectlibrary
REM It creates a relationship between Operation1 and Subassyl
REM Afterwards it replaces the participating object Operation1
of the relationship by Operation2.
```

```
sub main (parent_id)
```

```
    ' call this script on projects only
    subassyl = CreateComp (parent_id, "", "Subassembly", "Subassyl")
    operation1 = CreateComp (parent_id, "", "Operation", "Operation1")
    operation2 = CreateComp (parent_id, "", "Operation", "Operation2")
    ' create relationship (Operation1, "") -----
    process_implements_requirement -----> (Subassyl,
    "unique_id_for_req_1")
    relship1 = data.
    ta.CreateRelationshipEx("process_implements_requirement",
    operation1, subassyl, "", _
    "unique_id_for_req_1", "")
    relship1_info = data.GetAttributeById(relship1,
    "relationshipinfo")
    MsgBox "relationship successfully created: " & vbNewLine &
```



```

relship1_info
' change given relationship to
' create relationship (Operation2, "") -----
process_implements_requirement -----> (Subassy1,
"changed_id")
' bitmask of cmSourceExposed (1) and cmTargetUnexposed (8) ==> 9
call Data.ChangeRelationship(9, relship1, operation2, "", "",
"changed_id", "", "#DROP(R(2))")
relship1_info = data.GetAttributeById(relship1,
"relationshipinfo")
MsgBox "relationship successfully changed to: " & vbNewLine &
relship1_info
end sub

function CreateComp (parent_id, childlistname, childname,
namecomp)
comp = data.CreateComponent(parent_id, childlistname, childname)
data.SetAttributebyId comp, "name", namecomp
CreateComp = comp
end function

```



### Note

*Changes participating objects of a relationship. This method should be used to change one or more objects participating in a relationship. It takes care about consistency checks that would not be done if participating objects are changed using method SetAttribute(s).*

#### 5.1.1.41 ExecuteServerMethodEx

##### SYNTAX

**ExecuteServerMethodEx (bstrObjectId, bstrMethod, pVarArrayIn)**

Parameters	Description
bstrObjectId	The object ID for which the function must be executed.
bstrMethod	A string identifier describing the function (branch) to be executed.
pVarArrayIn	An array containing parameters.

### Example

#### Example

```

sub main(id)
rem 0, 0: CopyOptions (10 = EPCO_COPY_DEEP|EPCO_COPY_RIGHTS)
rem 0, 1..n, 1: All [n] objects which should be copied
dim inArray(0, 1)
inArray(0, 0) = 10
inArray(0, 1) = id

outArray = data.ExecuteServerMethodEx (id,
"key_copy_return_objects", inArray)

ubound1 = ubound(outArray)
for i = 0 to ubound1
    rem The only column: Copies
    MsgBox(outArray(i))
next
end sub

```



### Note

*This call triggers a special implementation in the server which might be specific for the object type of the passed object (ID). The passed function identifier (For example, "freqcalc") specifies the branch which have to be executed.*

#### 5.1.1.42 GetRelationshipsForOwnerByNameEx

**SYNTAX**

**GetRelationshipsForOwnerByNameEx (bFilter, bstrSourceObjectId, bstrOwnerObjectId, bstrRelationName)**

**Parameters**

Parameters	Description
bFilter	If bFilter is set to true, only objects that match the filter criteria are contained in the vRelationShipIds.
bstrSourceObjectId	The ID of the child object for which relationships to other children is to be retrieved.
bstrOwnerObjectId	The object ID of the base object that owns the graph.
bstrRelationName	The name of the relationship, for example, "process_runsbefore_process".

**Script Return Value**

Parameter	Description
vRelationShipIds	An array of object IDs representing all relationships between a distinct child and the other children of the graph owning base object.

**Example****Example**

```

Dim relationship_name
sub main(id)
relationship_name = "process_runsbefore_process"
parent_process_id = GetBase(id)
child_process_sci_id = Data.GetFirstChild(parent_process_id,
"nodes")
Do while child_process_sci_id <> ""
child_process_id = GetBase(child_process_sci_id)
child_process_name = GetName(child_process_id)
rel_ids = Data.GetRelationshipsForOwnerByNameEx(true,
child_process_id, parent_process_id, relationship_name)
dimx = ubound(rel_ids)
if dimx >= 0 then
for i=0 to dimx
other_process_id = Data.GetOtherRelatedObject(rel_ids(i),
child_process_id)
other_process_name = GetName(other_process_id)
MsgBox("<" & child_process_name & "> has a relation of type <" &
relationship_name & "> to <" & other_process_name & ">.")
next
else
MsgBox("<" & child_process_name & "> has no relations of type <" &
relationship_name & "> to other subprocesses.")
end if

child_process_sci_id = Data.GetNextChild(parent_process_id,
"nodes")
Loop
end sub
'*****
***
function GetBase(id)
GetBase = Data.GetAttributebyId(id, "relationobject2")
end function
'*****
***
function GetName(id)
GetName = Data.GetAttributebyId(id, "name")

```

```
end function
```

### 5.1.1.43 OpenProperties

#### SYNTAX

**OpenProperties (ObjectId)**

#### Example

#### Example

```
sub main (objectId)
  Comp_id = Data.GetAttributById (objectId, "ergocompbase")
  Call Data.OpenProperties (Comp_id)
end sub
```

#### Script Return Value

None



#### Note

*This method allows the user to invoke the properties dialog of an object.*

*“ObjectId” has to be passed to OpenProperties method, where “ObjectId” represents the object for which user wants to invoke the properties dialog.*



#### Caution

*When Properties dialog is invoked via script, the following buttons are not shown:*

- Switch Action
- New Version
- Next
- Previous

*All the above button functionality is associated with the browser component; where as, in script engine, browser information is not available. Hence, when properties dialog is invoked via script, these buttons are not shown.*

*The Relationship tabs are also not displayed in properties dialog, for the same reason mentioned above.*

*Invoking properties dialog via script is not supported for the following component types:*

- Script command
- Script action

Script command and Script action properties dialog has its own dialog class functionality, apart from standard properties dialog. Hence, invoking properties dialog for these components is not supported via script.

## 5.2 Class SkriptItemRights

The rights concept has been subjected to a major change in a release PE 5.12. With PE 5.12 the permissions, users, and groups are stored in the object database, hence you can access information using for instance standard Get/SetAttributebyId calls. Please take into account that you have to (ex-) change scripts using this interface.

Table 11: Class SkriptItem Rights Methods

Method	Description	Availability
<a href="#">GetUserLogin</a>	Retrieve the login of the current user	PE 5.6 – 5.11
<a href="#">GetUserFullName</a>	Retrieve the description (used as full name) of the current user.	PE 5.6 – 5.11
<a href="#">GetCurrentUser</a> (PE 5.12)	Retrieve the object ID of the currently logged in user.	<b>PE 5.12 and later</b>
<a href="#">Transfer</a>	Transfer object(type) rights between two objects(types)..	<b>PE 5.10 – 5.11</b>
<a href="#">Transfer</a> (PE 5.12 and later)	Transfer permissions between two objects or plantypes.	<b>PE 5.12 and later</b>
<a href="#">GetSingleRight</a>	Get rights information (mask) of a distinct object, type, or function.	<b>PE 5.10 – 5.11</b>
<a href="#">GetSingleRight</a> (PE 5.12 and later)	Get rights information (mask) of a distinct object.	<b>PE 5.12 and later</b>
<a href="#">SetSingleRight</a> (PE 5.12 and later)	Set rights information (mask) of a distinct object.	<b>PE 5.12 and later</b>
<a href="#">GetAllGroups</a>	Get all user groups.	<b>PE 5.10 – 5.11</b>
<a href="#">GetAllGroups</a> (PE 5.12 and later)	Get all user groups.	<b>PE 5.12 and later</b>
<a href="#">GetAllUsers</a>	Get all users.	<b>PE 5.10 – 5.11</b>
<a href="#">GetAllUsers</a> (PE 5.12 and later)	Get all users.	<b>PE 5.12 and later</b>
<a href="#">GetUserInfoById</a>	Retrieve user information by user ID.	<b>PE 5.10 – 5.11</b>
<a href="#">GetGroupInfoById</a>	Retrieve group information by group ID.	<b>PE 5.10 – 5.11</b>
<a href="#">GetUserMemberships</a>	Retrieve a user's membership(s).	<b>PE 5.10 – 5.11</b>
<a href="#">GetUserMemberships</a> (PE 5.12 and later)	Retrieve a user's membership(s).	<b>PE 5.12 and later</b>
<a href="#">GetGroupMembers</a>	Get members of a distinct group.	<b>PE 5.10 – 5.11</b>
<a href="#">GetGroupMembers</a> (PE 5.12 and later)	Get members of a distinct group.	<b>PE 5.12 and later</b>
<a href="#">AddRights</a>	Add rights to an object, type, or function.	<b>PE 5.10 – 5.11</b>
<a href="#">RemoveRights</a>	Remove rights from an object, type, or function.	<b>PE 5.10 – 5.11</b>
<a href="#">ConvertRightMaskToArray</a>	Convert rights information from a bit mask to an array.	PE 5.10 and later
<a href="#">Create</a>	Create a user, a group, or a function right.	<b>PE 5.13 and later</b>
<a href="#">GetFunctionRights</a>	Get all function rights.	<b>PE 5.13 and later</b>
<a href="#">GetRightSubjects</a>	Get an objects rights informations.	
<a href="#">ResetItem</a>	Reset the script item to its initial state.	PE 5.10 and later

## 5.2.1 List of all XScriptItemRights Methods

### 5.2.1.1 GetUserLogin

Function is not available for PE 5.12 and later not required. *Please refer to the [GetCurrentUser](#).*

#### SYNTAX

**GetUserLogin()**

**Script Return Value**

**Example**

The current users login string.

**Example**

```
REM GetUserInfo
REM Script is independent of entry ID
sub main(id)
    login=Rights.GetUserLogin
    username=Rights.GetUserFullName
    caption="Benutzer Info"
    message = "Login: " & login & vbCRLF & "Name: " & username
    call Dialog.MessageBox(caption, message)
end sub
```

**Note**

*The user info structure contains more attributes than accessible via user interface or script.*

**5.2.1.2 GetUserFullName****SYNTAX**

**GetUserFullName();**

**Script Return Value**

Parameter	Description
BstrValue	The current users description (currently used as full name)

**Example****Example**

```
REM GetUserInfo
REM Script is independent of entry ID
sub main(id)
    login=Rights.GetUserLogin
    username=Rights.GetUserFullName
    caption="User Info"
    message = "Login: " & login & vbCRLF & "Name: " & username
    call Dialog.MessageBox(caption, message)
end sub
```

**5.2.1.3 GetCurrentUser (PE 5.12)****SYNTAX**

**GetCurrentUser();**

**Script Return Value**

Parameter	Description
bstrUserId	The object ID of the current user.

**Example****Example**

```
Rem Script demonstrates usage of Rights.Transfer method
Rem valid for PE5.12 and later, 2003-09-28, DELMIA
Rem Prerequisites: None.
Rem Result: Displays login and description of the currently
logged in user.
sub main(notused)
    user_id = Rights.GetCurrentUser()
    if user_id <> "" then
        REM retrieve the login of the user (is stored in the
attribute "nameshort")
        nameshort = Data.GetAttributebyId(user_id, "nameshort")
        MsgBox(nameshort)
        REM retrieve the description of the user (is stored in the
attribute "note")
```



```

desc = Data. GetAttributebyId(user_id, "note")
MsgBox(desc )
end if
end sub

```

### Note

*Beginning with R12, this function replaces the former `GetUserLogin` and `GetFullName`.*

*The user info structure contains more attributes than accessible via user interface or script.*

#### 5.2.1.4 Transfer

The function parameters have been changed for PE 5.12 and subsequent versions, refer to Transfer (PE 5.12 and later).

#### SYNTAX

**Transfer(bstrSourceObjectId, bstrTargetObjectId, bstrRightsDataType)**

Parameters	Description
bstrSourceObjectId	The object ID or (plan)type GUID.
bstrTargetObjectId	The object ID or (plan)type GUID.
bstrRightsDataType	"Object" or "Type"

Please note that you have to pass the correct object or type IDs dependent on the parameter bstrRightsDataType.

bstrRightsDataType	Objects	IDs
Object	Normal data objects	POET object Ids (typical format: \$id\$(0:0-8504#0, 339))
Type	Plantypes	GUIDs of <b>slave</b> plantypes (typical format: 3e7e6500-5df0-4286-b83e-103685035c14)
Type	Configuration Types	Configuration GUIDs as retrieved using XScriptItemConfig access functions (typical format: 3e7e6500-5df0-4286-b83e-103685035c14)

### Example

#### Example

```

sub main(notused)
...
REM Example for transferring object rights
call Rights.Transfer(child1_base_id, child2_base_id, "Object")
...
end sub

```

#### 5.2.1.5 Transfer (PE 5.12 and later)

#### SYNTAX

**Transfer(bstrSourceObjectId, bstrTargetObjectId, bstrTransferRightMode)**

Parameters	Description
bstrSourceObjectId	The object ID of the source object.
bstrTargetObjectId	The object ID of the target object.
bstrTransferRightMode	"Clone", "Add", or "Remove" ( <i>Please refer to the <a href="#">Note</a></i> )

#### Script Return Value



None

**Note**

*Please note that you have to pass the correct object or type IDs dependent on the parameter `bstrRightsData` type.*

Parameter - <code>bstrTransferRightMode</code>	Description
Clone	Copy the entire right mask from source to target object.
Add	Bit by bit adding of permissions from source to target object.
Remove	Bit by bit removal of permissions from source to target object.

**Example****Example**

Rem Script demonstrates usage of `Rights.Transfer` method  
 Rem valid for PE5.12 and later  
 Rem Prerequisites: A simple structure with a single parent and two children nodes  
 Rem Add some permissions to the first child node and the run the script on the parent node.

Rem Result: Permissions are cloned, added or removed to and from the second child node.

**Sub main(parent\_id)**

```

    parent_base_id = getBase(parent_id)
    child1_id = Data.GetFirstChild(parent_base_id, "nodes")
    child1_base_id = getBase(child1_id)
    child2_id = Data.GetNextChild(parent_base_id, "nodes")
    child2_base_id = getBase(child2_id)
    If child1_base_id <> "" And child2_base_id <> "" Then
        comp1Name = GetName(child1_base_id)
        comp2Name = GetName(child2_base_id)
        'Define Edit Controls for InputBox (Parameter: control id,
        control type,
        'prompt, default value)

        Call Dialog.CreateInputControl("1", "EditString", "Child
node 1", comp1Name)
        Call Dialog.ModifyInputControl("1", "ReadOnly", True)
        Call Dialog.CreateInputControl("2", "EditString", "Child
node 2", comp2Name)
        Call Dialog.ModifyInputControl("2", "ReadOnly", True)
        Dim radio(2)
        radio(0) = "Clone"
        radio(1) = "Add"
        radio(2) = "Remove"
        Call Dialog.CreateInputControl("3", "RadioButtons", "Trans-
fer Mode", radio)
        'display InputBox caption "Your Data"
        ret = Dialog.InputBox("Transfer Permissions")
        If ret <> 0 Then
            'check content of edit controls by means of control id
            (see above)
            radioval = Dialog.GetInputControlValue("3")
            If radioval = 0 Then
                transfermode = "Clone"
            ElseIf radioval = 1 Then
                transfermode = "Add"
            Else
                transfermode = "Remove"
            End If

```

```

        Call transfer(child1_base_id, child2_base_id, comp1Name,
        comp2Name, transfermode)
    End If
Else
    message = "Dialog canceled."
End If

End Sub

Sub transfer(child1_base_id, child2_base_id, comp1Name, comp2Name,
transfermode)
    MsgBox ("Now transferring rights from <" & comp1Name & "> to <" &
comp2Name & ">." & vbCrLf & "Transfer Mode is <" & transfermode & ">.")
    Call Rights.transfer(child1_base_id, child2_base_id, "Remove")
End Sub

Function getBase(id)
    getBase = Data.GetAttributebyId(id, "ergocompbase")
End Function

Function GetName(id)
    Name = Data.GetAttributebyId(id, "name")
    GetName = Name
End Function

```

### 5.2.1.6 GetSingleRight

The function parameters have been changed for PE 5.12 and subsequent versions, *Please refer to the [GetSingleRight \(PE 5.12 and later\)](#).*

#### SYNTAX

**GetSingleRight(IUserId, blsGroup, bstrObjectId, bstrRightsDataType)**

Parameters	Description
IUserId	The user or group ID to check for.
blsGroup	A flag indicating if IUserId is a user ID (0/False) or a group ID (1/True).
bstrObjectId	The object ID, plan(type) GUID, or function right name to check for ( <i>Please refer to the <a href="#">Note</a></i> )
bstrRightsDataType	"Object", "Type", or "Function" ( <i>Please refer to the <a href="#">Note</a></i> ).

#### Script Return Value

Parameter	Description
IRightMask	An integer representing a bit mask ( <i>Please refer to the <a href="#">Note</a></i> ).

### Example

```

sub main(id)
base_id = Data.GetAttributebyId(id, "ergocompbase")
users = Rights.GetAllUsers()
dimusers = ubound(users, 1)
for i = 0 to dimusers
call dispObjectRights(base_id, users(i))
next
end sub
sub dispObjectRights(id, userid)
userInfo = Rights.GetUserInfoById(userid)
rmessage = "User: " & userinfo(0) & vbCrLf & "User-ID: " &
cstr(userid) & vbCrLf & vbCrLf
rvalue = Rights.GetSingleRight(userid, 0, id, "Object")
rinfo = Rights.ConvertRightMaskToArray(rvalue)
dimrinfo = ubound(rinfo, 1)
for i = 0 to dimrinfo

```



```

rmessage = rmessage & rinfo(i,0) & " --> " & cstr(rinfo(i,1)) &
vbCRLF
next
MsgBox(rmessage)
end sub

```



### Note

User and group IDs are simple integers. The IDs are independent. Please note that you have to pass the correct object or type IDs dependent on the parameter *bstrRightsDataType*.

Parameter - <i>bstrRightsDataType</i>	Objects	IDs
Object	Normal data objects	POET object IDs (typical format: \$id\$(0:0-8504#0, 339))
Type	Plantypes	GUIDs of <b>slave</b> plantypes (typical format: 3e7e6500-5df0-4286-b83e-103685035c14)
Type	Configuration Types	Configuration GUIDs as retrieved using XScriptItemConfig access functions (typical format: 3e7e6500-5df0-4286-b83e-103685035c14)
Function	Function rights	A string like "VBA/Open IDE".

#### 5.2.1.7 GetSingleRight (PE 5.12 and later)

##### SYNTAX

**GetSingleRight(bstrObjectId, bstrRightSubjectId);**

Parameters	Description
bstrObjectId	The object ID from which the permission are to be read.
bstrRightSubjectId	The object ID of the user or the group for which the permissions concerning <i>bstrObjectId</i> are to be read.

##### Script Return Value

Parameters	Description
iRightMask	An integer representing a bit mask ( <i>Please refer to the <a href="#">Note</a></i> ).

### Example

Rem Script demonstrates usage of Rights.GetAllUsers, Rights.GetAllGroups, Rights.GetSingleRight and Rights.ConvertRightMaskToArray methods  
 Rem valid for PE5.12 and later, 2003-09-28, DELMIA  
 Rem Prerequisites: A single object node  
 Rem Create some users and add permissions from them to the object node on which you would like to test the script.

```

Sub main(id)
  base_id = Data.GetAttributebyId(id, "ergocompbase")
  MsgBox("User Permissions")
  users = Rights.GetAllUsers()
  dimusers = UBound(users, 1)
  For i = 0 To dimusers
    Call dispObjectRights(base_id, users(i))
  Next

  MsgBox("Group Permissions")
  groups = Rights.GetAllGroups()
  dimgroups = UBound(groups, 1)
  For i = 0 To dimgroups

```

```

        Call dispObjectRights(base_id, groups(i))
    Next
End Sub

Sub dispObjectRights(id, userid)
    If userid <> "" Then
        profile = Data.GetAttributebyId(userid, "nameshort")
    End If
    rmessage = "User: " & profile & vbCrLf & "User-ID: " & userid
    & vbCrLf & vbCrLf
    rvalue = Rights.GetSingleRight(id, userid)
    rinfo = Rights.ConvertRightMaskToArray(rvalue)
    dimrinfo = UBound(rinfo, 1)
    For i = 0 To dimrinfo
        rmessage = rmessage & rinfo(i, 0) & " --> " & CStr(rinfo(i,
1)) & vbCrLf
    Next
    MsgBox (rmessage)
End Sub

```

**Note**

*If you pass an empty string for `bstrRightSubjectId` then the permissions for the currently logged in user are retrieved.*

**5.2.1.8 SetSingleRight (PE 5.12 and later)****SYNTAX**

**GetSingleRight(bstrObjectId, bstrRightSubjectId, IRightMask)**

Parameters	Description
BstrObjectId	The object ID on which the permissions are to be set.
bstrRightSubjectId	The object ID of the user or the group for which the permissions concerning <i>bstrObjectId</i> are to be set.
IRightMask	The rights bit mask representing the permissions to be set.

**Script Return Value**

None

**Example****Example**

```

sub main(id)
    base_id = GetBase(id)
    comp_name = Data.GetAttributebyId(base_id, "name")
    user_id = Rights.GetCurrentUser
    project_id = GetProject(base_id)
    project_name = Data.GetAttributebyId(project_id, "name")
    project_permissions = Rights.GetSingleRight(project_id,
user_id)
    MsgBox("Transferring the projects <" & project_name &
">permissions to node <" & comp_name & ">.")
    call Rights.SetSingleRight(base_id, user_id,
project_permissions)
end sub
function GetBase(id)
    GetBase = Data.GetAttributebyId(id, "ergocompbase")
end function
function GetProject(id)
    GetProject = Data.GetAttributebyId(id, "ergoproject")
end function

```

**Note**

If you pass an empty string for *bstrRightSubjectId* then the permissions for the currently logged in user are set.

### 5.2.1.9 GetAllGroups

The function parameters have been changed for PE 5.12 and subsequent versions, *Please refer to the [GetAllGroups \(PE 5.12 and later\)](#).*

#### Syntax

**GetAllGroups()**

**Script Return Value**

#### Example

Parameter	Description
pGroupIds	An array of integers representing all group IDs.

#### Example

```
sub main(notused)
...
groups = Rights.GetAllGroups
dimgroups = ubound(groups, 1)
for i = 0 to dimgroups
call GetGroupInfo(groups(i))
next
...
end sub
```

### 5.2.1.10 GetAllGroups (PE 5.12 and later)

#### Syntax

**GetAllGroups();**

**Script Return Value**

#### Example

Parameter	Description
pGroupIds	An array of integers representing all group object IDs.

#### Example

```
sub main(notused)
...
groups = Rights.GetAllGroups
dimgroups = UBound(groups, 1)
For i = 0 To dimgroups
Call dispObjectRights(base_id, groups(i))
Next
...
end sub
```

*Please refer to the [GetSingleRight \(PE 5.12 and later\)](#)*

### 5.2.1.11 GetAllUsers

The function parameters have been changed for PE 5.12 and subsequent versions, *Please refer to the [GetAllUsers \(PE 5.12 and later\)](#).*

#### Syntax

**GetAllUsers()**

**Script Return Value**

#### Example

Parameter	Description
pUserIds	An array of integers representing all user IDs.

#### Example

```
sub main(notused)
...
users = Rights.GetAllUsers
dimusers = ubound(users, 1)
```

```

for i = 0 to dimusers
call GetUserInfo(users(i))
next
...
end sub

```

### 5.2.1.12 GetAllUsers (PE 5.12 and later)

#### Syntax

#### GetAllUsers()

#### Script Return Value

Parameter	Description
pUserIds	An array of integers representing all user object IDs.

#### Example

#### Example

```

sub main(notused)
...
users = Rights.GetAllUsers
dimusers = ubound(users, 1)
for i = 0 to dimusers
call GetUserInfo(users(i))
next
...
end sub

```

Please refer to the [GetSingleRight \(PE 5.12 and later\)](#).

### 5.2.1.13 GetUserInfoById

Function is not available for PE 5.12 and later anymore (not required).

#### Syntax

#### GetUserInfoById(IUserId)

Parameter	Description
IUserId	The user ID.

#### Script Return Value

Parameter	Description
pUserInfo	An 1-dim array (pair) user name (login)/user full name (description).

#### Example

#### Example

```

...
sub GetUserInfo(userid)
userInfo = Rights.GetUserInfoById(userid)
dimuserinfo = ubound(userInfo, 1)
if dimuserinfo < 0 then
rmessage = "User does not exist (invalid user ID " & userid & "
)"
else
rmessage = "User: " & userinfo(0) & vbTab & "User-ID: " &
cstr(userid)
end if
ts.WriteLine(rmessage)
end sub
...

```

### 5.2.1.14 GetGroupInfoById

Function is not available for PE 5.12 and later anymore (not required).

#### Syntax

#### GetGroupInfoById(IGroupId);

Parameter	Description
-----------	-------------

Parameter	Description
lGroupId	The group ID.

**Script Return Value**

Parameter	Description
pGroupInfo	An 1-dim array (pair) group name/group description.

**Example****Example**

```
...
sub GetGroupInfo(groupId)
groupInfo = Rights.GetGroupInfoById(groupId)
dimgroupinfo = ubound(groupInfo, 1)
if dimgroupinfo < 0 then
rmessage = "Group does not exist (invalid group ID " & groupId &
" )"
else
rmessage = "Group: " & groupInfo(0) & vbTab & "Group-ID: " &
cstr(groupId)
end if
ts.WriteLine(rmessage)
end sub
...
```

**5.2.1.15 GetUserMemberships**

The function parameters have been changed for PE 5.12 and subsequent versions, *Please refer to the [GetUserMemberships \(PE 5.12 and later\)](#).*

**Syntax****GetUserMemberships(IUserId)**

Parameter	Description
IUserId	The user ID.

**Script Return Value**

Parameter	Description
pGroupIds	An array of integers representing all group IDs to which the user IUserId belongs.

**Example****Example**

```
sub main(notused)
...
usermemberships=Rights.GetUserMemberships(users(i))
dimusermemberships = ubound(usermemberships, 1)
if dimusermemberships < 0 then
ts.WriteLine("Not a member of a special group.")
end if
for j = 0 to dimusermemberships
call GetGroupInfo(usermemberships(j))
next
...
end sub
```

**5.2.1.16 GetUserMemberships (PE 5.12 and later)****Syntax****GetUserMemberships(bstrUserObjectId)**

Parameter	Description
bstrUserObjectId	The user object ID.

**Script Return Value**

Parameter	Description
-----------	-------------

Parameter	Description
pGroupIds	An array of strings representing all group object IDs to which the user bstrUserObjectId belongs.

**Example****Example**

```
sub main(notused)
...
usermemberships=Rights.GetUserMemberships(users(i))
dimusermemberships = UBound(usermemberships, 1)
next
...
end sub
```

**Note**

The function call syntax has only slightly changed compared to R11. However, the passed parameter is an object ID string and the returned array consists of object ID strings (POET typical format: "\$id\$(0:0-8504#0, 339)" ) - instead of the former used unsigned integers (0, 1, ..., n).

**5.2.1.17 GetGroupMembers**

The function parameters have been changed for PE 5.12 and subsequent versions, Please refer to the [GetGroupMembers \(PE 5.12 and later\)](#).

**Syntax****GetGroupMembers(IGroupId)**

Parameter	Description
IGroupId	The group ID.

**Script Return Value**

Parameter	Description
pUserIds	An array of integers representing all user IDs belonging to the group having ID IGroupId.

**Example****Example**

```
sub main(notused)
...
groupmembers = Rights.GetGroupMembers(groups(i))
dimgroupmembers = ubound(groupmembers, 1)
if dimgroupmembers < 0 then
ts.WriteLine("No members in this group.")
end if
for j = 0 to dimgroupmembers
call GetUserInfo(groupmembers(j))
next
...
end sub
```

**5.2.1.18 GetGroupMembers (PE 5.12 and later)****Syntax****GetGroupMembers(bstrGroupObjectId);**

Parameter	Description
bstrGroupObjectId	The group object ID.

**Script Return Value**

Parameter	Description
pUserIds	An array of strings representing all user object IDs belonging to the group having group object ID <i>bstrGroupObjectId</i> .

**Example****Example**

```
Rem Script demonstrates usage of Rights.Transfer method
Rem valid for PE5.12 and later
Rem Prerequisites: At least a few users and some groups to which
the users are related.
Rem Result: A text file which displays the users and groups
contained in the database as well as their members and
memberships, respectively.
Dim fso, ts

Sub main(notused)
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set ts = fso.CreateTextFile("C:\Temp\UserGroupMatrix.txt",
True)
    users = Rights.GetAllUsers
    dimusers = UBound(users, 1)
    ts.WriteLine ("***All Users")
    For i = 0 To dimusers
        Call GetUserInfo(users(i))
    Next
    ts.WriteLine ("")
    groups = Rights.GetAllGroups
    dimgroups = UBound(groups, 1)

    ts.WriteLine ("***All Groups")
    For i = 0 To dimgroups
        Call GetGroupInfo(groups(i))
    Next
    ts.WriteLine ("")

    ts.WriteLine ("***All Group Members")
    For i = 0 To dimgroups
        Call GetGroupInfo(groups(i))
        groupmembers = Rights.GetGroupMembers(groups(i))
        dimgroupmembers = UBound(groupmembers, 1)
        If dimgroupmembers < 0 Then
            ts.WriteLine ("No members in this group.")
        End If
        For j = 0 To dimgroupmembers
            Call GetUserInfo(groupmembers(j))
        Next
        ts.WriteLine ("")
    Next
    ts.WriteLine ("")

    ts.WriteLine ("***All User Memberships")
    For i = 0 To dimusers
        Call GetUserInfo(users(i))
        usermemberships = Rights.GetUserMemberships(users(i))
        dimusermemberships = UBound(usermemberships, 1)
        If dimusermemberships < 0 Then
            ts.WriteLine ("Not a member of a special group.")
        End If
        For j = 0 To dimusermemberships
            Call GetGroupInfo(usermemberships(j))
        Next
        ts.WriteLine ("")
    Next
    MsgBox ("Done.")
    ts.Close
End Sub
```

```

Sub GetUserInfo(userid)
    If userid <> "" Then
        Name = Data.GetAttributebyId(userid, "nameshort")
        desc = Data.GetAttributebyId(userid, "note")
        rmessage = "User ID: " & userid & " >> " & Name & " >> " &
desc
        ts.WriteLine (rmessage)
    End If
End Sub

Sub GetGroupInfo(groupid)
    If groupid <> "" Then
        Name = Data.GetAttributebyId(groupid, "nameshort")
        desc = Data.GetAttributebyId(groupid, "note")
        rmessage = "Group ID: " & groupid & " >> " & Name & " >> "
& desc
        ts.WriteLine (rmessage)
    End If
End Sub

```



### Note

*The function call syntax has only slightly changed compared to PE 5.11. However, the passed parameter is an object ID string and the returned array consists of object ID strings (POET typical format: "\$id\$(0:0-8504#0, 339)" ) - instead of the former used unsigned integers (0, 1, ..., n).*

#### 5.2.1.19 AddRights

Function is not available for PE 5.12 and later anymore. Use [SetSingleRight \(PE 5.12 and later\)](#) instead.

#### Syntax

**AddRights(bstrObjectId, bstrRightsDataType, pRightsInfo)**

Parameters	Description
bstrObjectId	Object ID, or (plan)type GUID of the object to be changed.
bstrRightsDataType	"Object" (for objects) or "Type" (for plantypes and configuration types)
pRightsInfo	A structure containing the rights information to add. ( <i>Please refer to the <a href="#">Note</a>.</i> )

#### Example

#### Example

```

sub main(id)
base_id = Data.GetAttributebyId(id, "ergocompbase")
userid = 5
rightmask = 782 ("Change")
Dim rightinfo(0,3)
rightinfo(0,0) = userid
rightinfo(0,1) = 0
rightinfo(0,2) = rightmask
rightinfo(0,3) = 0

call Rights.AddRights(base_id, "Object", rightinfo)
...
end sub

```

#### 5.2.1.20 RemoveRights

Function is not available for PE 5.12 and later anymore. Use [SetSingleRight \(PE 5.12 and later\)](#) instead.

#### Syntax

**RemoveRights([bstrObjectId, bstrRightsDataType, pRightsInfo)**



Parameters	Description
bstrObjectId	Object ID, or (plan)type GUID of the object to be changed.
bstrRightsDataType	"Object" (for objects) or "Type" (for plantypes and configuration types)
pRightsInfo	A structure containing the rights information to add ( <i>Please refer to the <a href="#">Note</a></i> ).

**Script Return Value**

None

**Example****Example**

```

sub main(id)
base_id = Data.GetAttributebyId(id, "ergocompbase")
userid = 5
rightmask = 782 ("Change")

Dim rightinfo(0,3)
rightinfo(0,0) = userid
rightinfo(0,1) = 0
rightinfo(0,2) = rightmask
rightinfo(0,3) = 0

call Rights.RemoveRights(base_id, "Object", rightinfo)
...
end sub

```

**Note**

The rights info structure is an ( $n \times 4$ ) dimensional array.  $n$  means that you can realize changes for an arbitrary number of users and groups in a single call. The members (0,0) - (0,3) represent the first user (group), the members (1,0) - (1,3) the second user (group), etc...

The structure is organized as follows

Element	Contents	Note
0	User ID	
1	IsGroup Flag	0: User 1: Group
2	Right Mask	An integer representing the setting of the right bit mask
3	IsOwner Flag	0: User isn't owner of the object. 1: User is owner of the object.

**5.2.1.21 ConvertRightMaskToArray****Syntax**

**ConvertRightMaskToArray(IRightMask);**

Parameter	Description
IRightMask*	Edit

**Script Return Value**

Parameter	Description
*pRightsArray	Edit

**Example****Example (runs in PE 5.10-11 only)**

```

sub main(id)
base_id = Data.GetAttributebyId(id, "ergocompbase")

```

```

users = Rights.GetAllUsers()
dimusers = ubound(users, 1)
for i = 0 to dimusers
    call dispObjectRights(base_id, users(i))
next
end sub
sub dispObjectRights(id, userid)
    userInfo = Rights.GetUserInfoById(userid)
    rmessage = "User: " & userInfo(0) & vbCRLF & "User-ID: " &
cstr(userid) & vbCRLF & vbCRLF
    rvalue = Rights.GetSingleRight(userid, 0, id, "Object")
    rinfo = Rights.ConvertRightMaskToArray(rvalue)
    dimrinfo = ubound(rinfo, 1)
    for i = 0 to dimrinfo
        rmessage = rmessage & rinfo(i,0) & " --> " &
cstr(rinfo(i,1)) & vbCRLF
    next
    MsgBox(rmessage)
end sub

```

**Example PE 5.12**

Please refer to the [GetSingleRight \(PE 5.12 and later\)](#)

**Note**

\*The right mask is organized as follows

Bit	9	8	7	6	5	4	3	2	1	0	Decimal
"Atomic" Right	Remove Child	Add Child	Change Rights	Take Ownership	Erase	Create	Change	Execute	Read	No Access	
Object Composite Right											
Full Access	1	1	1	1	1	0	1	1	1	1	1007
Write	1	1	0	0	1	1	1	1	1	0	830
Change	1	1	0	0	0	0	1	1	1	0	782
No rights defined (rights are inherited)	0	0	0	0	0	0	0	0	0	0	0

**5.2.1.22 Create****Syntax**

**Create(bstrObjectType);**

Parameter	Description
bstrObjectType	The type that specifies what to create. Valid strings are "User", "Group", or "Function"

**Script Return Value**

Parameter	Description
bstrRightsObjectId	The object that has been created. This is either a user, a group, or a

Parameter	Description
	function right.

**Example****Example**

```
sub main(notused)
...
new_user_id = Rights.Create("User")
if new_user_id <> "" then

call Data.SetAttributeById(new_user_id, "nameshort", "FGB")
call Data.SetAttributeById(new_user_id, "externalid", "FGB")
call Data.SetAttributeById(new_user_id, "note", "Frank
Gugenberger")
'R14 and previous versions
'call Data.SetAttributeById(new_user_id, "password", "fgb")
'R15 and later versions - modification of script example
required before use! Refer to remarks.
'call Data.SetAttributeById(new_user_id, "password",
"c05651bcf32bd68259c0862d9f10cea9b0c14bcc")
...
end sub
```

**Note**

*Creating a user, a group, or a function right requires proper initialization. Otherwise, the object might not be accessible from the user interface. If you are not sure about the correct handling, do not use the function and ask for support.*

Password Encryption (case-sensitive):

- 1) Open a command window and enter the following command line  
**C:\temp> simplecryptadmin -hs <String> SHA1**  
 (<String>: Password string to be encrypted - without quotation marks)  
 In the example above this would read:  
**C:\temp> simplecryptadmin -hs fgb SHA1**  
 The result will look like this:  
 String Hash: c05651bcf32bd68259c0862d9f10cea9b0c14bcc  
 Operation successfully done
- 2) In the script replace the password in line  
**call Data.SetAttributeById(new\_user\_id, "password", "fgb")**  
 by the encrypted password string as generated in step 1  
**call Data.SetAttributeById(new\_user\_id, "password",  
 "c05651bcf32bd68259c0862d9f10cea9b0c14bcc"**

**5.2.1.23 GetFunctionRights****Syntax**

**GetFunctionRights();**

Parameter	Description
none	-

**Script Return Value**

Parameter	Description
pFunctionIds	An array containing all function right objects.

**Note**

The returned function right array is flat, but contains hierarchical information, because function rights are organized in a tree structure. Each function right has an attribute "rootfunctionname". If this attribute is empty, then it is a top-level function right. Otherwise it contains the object ID of this function rights "parent".

**Example****Example**

```
sub main(notused)

users = Rights.GetAllUsers()
dimusers = ubound(users, 1)

functionname="Copy Project"

for i = 0 to dimusers
    call dispFunctionRights(functionname, users(i))
next

end sub

sub dispFunctionRights(id, userid)
    userInfo = Rights.GetUserInfoById(userid)
    rmessage = "User: " & userInfo(0) & vbCRLF & "User-ID: " &
    cstr(userid) & vbCRLF & vbCRLF
    rvalue = Rights.GetSingleRight(userid, 0, id, "Function")
    rinfo = Rights.ConvertRightMaskToArray(rvalue)
    dimrinfo = ubound(rinfo, 1)
    for i = 0 to dimrinfo
        rmessage = rmessage & rinfo(i,0) & " --> " _
        & cstr(rinfo(i,1)) & vbCRLF
    next
    MsgBox(rmessage)
end sub
```

**5.2.1.24 GetRightSubjects****Syntax****GetRightSubjects(bstrObjectId)**

Parameter	Description
bstrObjectId	The object ID from which the permission are to be read.

**Script Return Value**

Parameter	Description
IRightMask	An integer representing a bit mask (see Note).

**Note**

If you pass an empty string for bstrRightSubjectId then the permissions for the currently logged in user are retrieved.

**Example****Example**

```
sub main(sci_id)
    base_id = Data.GetAttributeById(sci_id, "ergocompbase")
    ids = Rights.GetRightSubjects(base_id)
    dim_ids = ubound(ids, 1)
    name = Data.GetAttributeById(ids(0), "name")
    msgbox(dim_ids & " " & ids(0) & " " & name)
end sub
```

## 5.3 Class XScriptItemGrid

The [Table 12](#) demonstrates the class scriptitem grid methods.

**Table 12: Class XScriptItem Grid Methods**

Method	Description
Create	Create a sheet (grid).
SetCell	Set the value of a distinct cell in the grid.
ResetItem	Reset the script item to its initial state.

### 5.3.1 List of XScriptItemGrid Methods

#### 5.3.1.1 Create

##### SYNTAX

**Create(bstrCaption, ulRows, ulCols, bstrColumnHeaders)**

Parameters	Description
bstrCaption	The caption of the list view containing the grid.
ulRows	The number of rows of the grid.
ulCols	The number of columns of the grid.
bstrColumnHeaders	The column headers. Pass the headers as a single string separating the entries by a ' ' (pipe) character.

##### Script Return Value

None

##### Example

#### Example

```
REM Display a sheet
REM skript is independent of entry ID
sub main (id)
    call Sheet.Create("My Sheet", 10, 3, "Col1|Col2|Col3")
    call Sheet.SetCell(1,1, "1000")
    call Sheet.SetCell(5,1, "5000")
end sub
```



##### Note

*It is necessary to check the flag "Display sheet" in the properties dialog of the script to initialize a client window that holds the grid.  
At present the XScriptItemGrid class is for viewing purpose only and have only rudimentary functionality.*

#### 5.3.1.2 SetCell

##### SYNTAX

**SetCell(ulRow, ulCol, vValue,)**

Parameters	Description
ulRow	The row index of the cell. Uppermost is 1.
ulCol	The row index of the cell. Leftmost is 1.
vValue	The value to be displayed in the cell.

**Script Return Value**

None

**Example****Example**

```
REM Display a sheet
REM skript is independent of entry ID
sub main (id)
    call Sheet.Create("My Sheet", 10, 3, "Col1|Col2|Col3")
    call Sheet.SetCell(1,1, "1000")
    call Sheet.SetCell(5,1, "5000")
end sub
```

**Note**

*It is necessary to check the flag "Display sheet" in the properties dialog of the script to initialize a client window that holds the grid. At present the XScriptItemGrid class is for viewing purpose only and have only rudimentary functionality.*

## 5.4 Class ScriptItemGraphic

The [Table 13](#) demonstrates the class scriptitem graphic methods.

**Table 13: Class Scriptitem Graphic Methods**

Method	Description
<a href="#">CreateBitmap</a>	Create a bitmap of the associated 3D graphic scene.
<a href="#">CreateContextBitmap</a>	Create a bitmap of the associated 3D graphic scene in its context (the graphic itself and an overlap to the neighboring elements).
<a href="#">CreateFile</a>	Create an image file (*.bom or *.objekt format) of a graphic object.
<a href="#">GetBBoxSize</a>	Get the bounding box of the associated 3D graphic element.
<a href="#">GetGraphicMatrix</a>	Get the 4x4 graphic matrix (rotations and translations).
<a href="#">GetGraphicRGB</a>	Get the RGB color values of a graphic object.
<a href="#">GetGraphicAlpha</a>	Get the alpha value of a graphic object.
<a href="#">GetTranslation</a>	Get the translation of a graphic object.
<a href="#">GetRotation</a>	Get the rotation of a graphic object.
<a href="#">SetGraphicMatrix</a>	Set the 4x4 graphic matrix (rotations and translations).
<a href="#">SetGraphicRGB</a>	Set the RGB color values of a graphic object.
<a href="#">SetGraphicAlpha</a>	Set the alpha value of a graphic object.
<a href="#">SetTranslation</a>	Set the translation of a graphic object.
<a href="#">SetRotation</a>	Set the rotation of a graphic object.
<a href="#">CreatePreviewGraphic</a>	Graphic files are created and saved in a special directory, because the DPE graphical formats are not readable with DPM.
<a href="#">ShowGraphic</a>	Method for the creation of a graphic view.
<a href="#">ResetItem</a>	Reset the script item to its initial state.

### 5.4.1 List of XScriptItemGraphic Methods

#### 5.4.1.1 CreateBitmap SYNTAX

**CreateBitmap(bstrObjectId, bstrFileName, bstrSettings, dblSizeA, dblSizeB)**

Parameters	Description
bstrObjectId	A string containing the ID of the object.
bstrFileName	The name of file to which the bitmap should be saved.
bstrSettings	Define the graphic settings.
dblSizeA	The width of the bitmap. Values are rounded to an integer.
dblSizeB	The height of the bitmap. Values are rounded to an integer.

**Script Return Value**

None

**Example****Example (An Extract)**

```

sub main(id)
...
  camposx=0.0
  camposy= round(((ymax(i)+ymin(i)) / 2), 0)
  camposz= round(((zabsmax + zabsmin) / 2), 0)

  width=0.0
  height = zabsmax - zabsmin
  scale = 768/height

  bmpsizey=scale * (ymax(i)-ymin(i))
  bmpsizez=768
  settings="[VIEW:Left Side View][CAMPAR2D:" & camposx & "|" &
camposy & "|" & camposz & "|" & width & "|" & height & "]"
  filename = filename_part + "_LeftSideView" + fileextension

  call Graphic.CreateBitmap(child_id, filename, settings,
bmpsizey, bmpsizez)
...
end sub

```

**Note**

*If you pass bstrSettings="" then the bitmaps contains a screenshot of the default 3D view.*

**5.4.1.2 CreateContextBitmap****SYNTAX****CreateContextBitmap(bstrObjectId, bstrContextObjectId, bstrFileName, bstrSettings, dblSizeA, dblSizeB)**

Parameters	Description
bstrObjectId	A string containing the ID of the object.
bstrContextObjectId	A string containing the ID of the context object.
bstrFileName	The name of file to which the bitmap should be saved.
bstrSettings	Define the graphic settings.
dblSizeA	The width of the bitmap. Values are rounded to an integer.
dblSizeB	The height of the bitmap. Values are rounded to an integer.

**Script Return Value**

None

**Example****Example (An Extract)**

```

sub main(id)

```

```

...
camposx=0.0
camposy= round(((ymax(i)+ymin(i)) / 2), 0)
camposz= round(((zabsmax + zabsmin) / 2), 0)
width=0.0
height = zabsmax - zabsmin
scale = 768/height
bmpsizey=scale * (ymax(i)-ymin(i))
bmpsizez=768
settings="[VIEW:Left Side View][CAMPAR2D:" & camposx & "|" &
camposy & "|" & camposz & "|" & width & "|" & height & "]"
filename = filename_part + "_LeftSideView" + fileextension

call Graphic.CreateContextBitmap(child_id, parent_id, filename,
settings, bmpsizey, bmpsizez)
...
end sub

```



### Note

*If you pass bstrSettings="" then the bitmaps contains a screenshot of the default 3D view.*

#### 5.4.1.3 GetBBoxSize

##### SYNTAX

**GetBBoxSize(bstrObjectId);**

Parameter	Description
bstrObjectId	A string containing the ID of the object.

##### Script Return Value

An array of the 3x2 coordinates giving the position of the bounding box.

The order is as follows:

0	1		X <sub>min</sub>	X <sub>max</sub>
2	3	=	Y <sub>min</sub>	Y <sub>max</sub>
4	5		Z <sub>min</sub>	Z <sub>max</sub>

### Example

##### Example

```

sub main (id)
...
bboxsize = Graphic.GetBBoxSize(child_id)
xmin(i)=round(bboxsize(0),0)
xmax(i)=round(bboxsize(1),0)
ymin(i)=round(bboxsize(2),0)
ymax(i)=round(bboxsize(3),0)
zmin(i)=round(bboxsize(4),0)
zmax(i)=round(bboxsize(5),0)
...
end sub

```



### Note

*The bounding box is the smallest cuboid a graphic object fits in completely.*

#### 5.4.1.4 CreateFile

##### SYNTAX

**CreateFile(bstrObjectId, bstrFileNameWithNoExtension)**



Parameters	Description
bstrObjectId	A string containing the ID of the object.
bstrFileNameWithNoExtension	<i>Please refer to the <a href="#">Table 14</a>.</i>

You can either pass an identifier and a file name without extension or a standard file extension which can be interpreted. *Please refer to the [Table 14](#) and [Example](#).*

**Table 14: Identifier and File Names**

Identifier	Description
b	Creates an DPE graphic file with extension .bom
o	Creates an old-style E4 graphic file with extension .objekt Do not use this format, except you like to re-import a graphic file into ERGOPLAN 4.
d	Creates an AutoCAD (DXF) export graphic file with extension .dxf
v	Creates a VRML1 graphic file with extension .wrl
c	Creates a Catia Tessellated Format (CGR) graphic file with extension .cgr

If no directory is specified then the default directory (xboggraph\_koerper) is used.

### Script Return Value

None

### Example

```
sub main(id)
path="c:\temp\"
name="mygraphic"
filename=path+name
call Graphic.CreateFile(id, "b " + filename)
' or using different notation
' call Graphic.CreateFile(id, filename + ".bom")
call Graphic.CreateFile(id, "d " + filename)
' or using different notation
' call Graphic.CreateFile(id, filename + ".dxf")
call Graphic.CreateFile(id, "v " + filename)
' or using different notation
' call Graphic.CreateFile(id, filename + ".wrl")
call Graphic.CreateFile(id, "c " + filename)
' or using different notation
' call Graphic.CreateFile(id, filename + ".cgr")
end sub
```

#### 5.4.1.5 GetGraphicMatrix

##### SYNTAX

**GetGraphicMatrix(BSTR bstrObjectId);**

Parameter	Description
bstrObjectId	A string containing the ID of the object.

### Script Return Value

An array of 16 double values gives the rotation/translation matrix of the associated graphic object.

r <sub>11</sub>	r <sub>12</sub>	r <sub>13</sub>	0	0	1	2	3
-----------------	-----------------	-----------------	---	---	---	---	---

r <sub>21</sub>	r <sub>22</sub>	r <sub>23</sub>	0	4	5	6	7
r <sub>31</sub>	r <sub>32</sub>	r <sub>33</sub>	0	8	9	10	11
t <sub>1</sub>	t <sub>2</sub>	t <sub>3</sub>	1	12	13	14	15

**Example****Example**

```
REM manipulate the position of a graphic object (translation only)
sub main (id)
  Dim matrix
  matrix = Graphic.GetGraphicMatrix(id)
  x=matrix(12)
  y=matrix(13)
  z=matrix(14)
  matrix(12) = x+1000
  matrix(13) = y+1000
  matrix(14) = z+1000
  call Graphic.SetGraphicMatrix(id, matrix)
end sub
```

**5.4.1.6 GetGraphicRGB****SYNTAX****GetGraphicRGB(bstrObjectId)**

Parameter	Description
bstrObjectId	A string containing the ID of the object.

**Script Return Value**

An array of three double values gives the RGB value of the associated graphic object.

R	0
G	1
B	2

**Example****Example**

```
sub main (id)
  Dim rgb
  rgb = Graphic.GetGraphicRGB(id)
  r=rgb(0)
  g=rgb(1)
  b=rgb(2)

  rgb(0)=100
  rgb(1)=100
  rgb(2)=100
  Call Graphic.SetGraphicRGB(id, rgb)
end sub
```

**5.4.1.7 GetGraphicAlpha****SYNTAX****GetGraphicAlpha(BSTR bstrObjectId)**

Parameter	Description
bstrObjectId	A string containing the ID of the object.

**Script Return Value**

The "alpha" (inverse transparency) of the object.

**Example**

**Example**

```
sub main (id)
  alpha = Graphic.GetGraphicAlpha(id)
  alpha = alpha/2
  call Graphic.SetGraphicAlpha(id, alpha)
end sub
```

**5.4.1.8 GetTranslation****SYNTAX****GetTranslation(BSTR bstrObjectId)**

Parameter	Description
bstrObjectId	A string containing the ID of the object.

**Script Return Value**

An array of three double values gives the translation (origin based) of the associated graphic object.

x	0
y	1
z	2

**Example****Example**

```
sub main (id)
  Dim position
  position = Graphic.GetTranslation(id)
  x=position(0)
  y=position(1)
  z=position(2)
  position(0)=500
  position(1)=1500
  position(2)=2500
  call Graphic.SetTranslation(id, position)
end sub
```

**5.4.1.9 GetRotation****SYNTAX****GetRotation(BSTR bstrObjectId)**

Parameter	Description
bstrObjectId	A string containing the ID of the object.

**Script Return Value**

An array of three double values (radians) gives the rotation (origin based) of the associated graphic object.

x	0
y	1
z	2

**Example****Example**

```
sub main (id)
  Dim angle
  angle = Graphic.GetRotation(id)
  x=angle(0)
  y=angle(1)
  z=angle(2)
  angle(0)=1.5
  angle(1)=1.5
  angle(2)=1.5
  call Graphic.SetRotation(id, angle)
```

```
end sub
```

### 5.4.1.10 SetGraphicMatrix

#### SYNTAX

**SetGraphicMatrix(BSTR bstrObjectId,pMatrix)**

Parameter				Description							
bstrObjectId				A string containing the ID of the object.							
pMatrix				An array of 16 double values gives the rotation/translation matrix of the associated graphic object.							
r <sub>11</sub>	r <sub>12</sub>	r <sub>13</sub>	0	0	1	2	3				
r <sub>21</sub>	r <sub>22</sub>	r <sub>23</sub>	0	4	5	6	7				
r <sub>31</sub>	r <sub>32</sub>	r <sub>33</sub>	0	8	9	10	11				
t <sub>1</sub>	t <sub>2</sub>	t <sub>3</sub>	1	12	13	14	15				

#### Script Return Value

None

#### Example

#### Example

```
REM manipulate the position of a graphic object (translation only)
sub main (id)
    Dim matrix
    matrix = Graphic.GetGraphicMatrix(id)

    x=matrix(12)
    y=matrix(13)
    z=matrix(14)

    matrix(12) = x+1000
    matrix(13) = y+1000
    matrix(14) = z+1000
    call Graphic.SetGraphicMatrix(id, matrix)
end sub
```



#### Note

*Be extremely careful when manipulating the rotation part of the matrix! It is recommended rather to use SetRotation instead.*

### 5.4.1.11 SetGraphicRGB

#### SYNTAX

**SetGraphicRGB(bstrObjectId, pRGB);**

Parameters		Description	
bstrObjectId		A string containing the ID of the object.	
pRGB		An array of three double values gives the RGB value of the associated graphic object (see below).	
R	0		
G	1		
B	2		

#### Script Return Value

None

**Example****Example**

```

sub main (id)
    Dim rgb
    rgb = Graphic.GetGraphicRGB(id)
    r=rgb(0)
    g=rgb(1)
    b=rgb(2)

    rgb(0)=100
    rgb(1)=100
    rgb(2)=100
    call Graphic.SetGraphicRGB(id, rgb)
end sub

```

**5.4.1.12 SetGraphicAlpha****SYNTAX**

**SetGraphicAlpha(bstrObjectId, dblAlpha);**

Parameters	Description
bstrObjectId	A string containing the ID of the object.
dblAlpha	The new value of "alpha" (inverse transparency) to set.

**Script Return Value**

None

**Example****Example**

```

sub main (id)
    alpha = Graphic.GetGraphicAlpha(id)
    alpha = alpha/2
    call Graphic.SetGraphicAlpha(id, alpha)
end sub

```

**5.4.1.13 SetTranslation****SYNTAX**

**SetTranslation(bstrObjectId, dblPosition);**

Parameters	Description
bstrObjectId	A string containing the ID of the object.
dblPosition	An array of three double values setting the new values of the translation (origin based).

x	0
y	1
z	2

**Script Return Value**

None

**Example****Example**

```

sub main (id)
    Dim position
    position = Graphic.GetTranslation(id)
    x=position(0)
    y=position(1)
    z=position(2)
    position(0)=500
    position(1)=1500
    position(2)=2500
    call Graphic.SetTranslation(id, position)

```

```
end sub
```

### 5.4.1.14 SetRotation

#### SYNTAX

**SetRotation(bstrObjectId, dblParameter)**

Parameters		Description
bstrObjectId		A string containing the ID of the object.
dblParameter	x	An array of three double values (radians) setting the new values of rotation (origin based).
	y	
	z	

#### Script Return Value

None

#### Example

#### Example

```
sub main (id)
  Dim angle
  angle = Graphic.GetRotation(id)
  x=angle(0)
  y=angle(1)
  z=angle(2)
  angle(0)=1.5
  angle(1)=1.5
  angle(2)=1.5
  call Graphic.SetRotation(id, angle)
end sub
```

### 5.4.1.15 CreatePreviewGraphic

#### SYNTAX

**CreatePreviewGraphic (bstrObjectId, bstrType, dblDeep, dblOverwrite, bUseExisting)**

Parameters	Description
bstrObjectId	A string containing the ID of the object.
bstrType	A string, indicating the graphic type of the created file. Supported types are: CGR:"cgr!", VRML "wrl", and simple DXF: "dxf".
dblDeep	Boolean. a boolean value indicating whether the operation should be executed recursively. In this case a preview file is created not only for the object (0), but for all of its children (1).
dblOverwrite	A boolean value indicating whether existing files is overwritten or not.
bUseExisting	If true the content of the graphicname attribute is copied to the previewgraphic attribute, if the referenced graphic file is of the same type as the target type. The file(s) is/are created; the attribute is set. Settings for file type and deep option are not used here.

#### Script Return Value

None

#### Example

#### Example

```
sub main (id)
  bstrType = "cgr"
  deep = 0
  overwrite = 1
  useexisting = 0
  call Graphic.CreatePreviewGraphic (id, type, deep, overwrite,
existing)
end sub
```

The directory on which the files are stored is specified with a new settings entry. This value is stored in the database, as a global setting under graphic. The directory name is DELMIA\PPRClient\preview\_graphic.

If the directory is not set in the PreviewGraphic then the cadpath and then product\_cadpth setting is used, for resources and products respectively. If none of them are no file gets created.

The functionality is available for any object with a graphicdata interface.

The functionality is restricted to admin users. A file name is generated automatically, using the GUID of the object.



### Note

*DPE does not use the created files. There are exclusively for DPM.*

- *Performance might be a problem; depending on the size of the involved geometries it takes a long time to create a set of preview geometries.*
- *Objects must not be locked by other user, because attribute must be set.*
- *Objects must be locked by user who uses the functionality, because attribute must be set*
- *Created files might not be a up to date, because there is no mechanism to delete, or automatically update the files.*
- *Created files might be misleading because of the option to use "graphicname" instead of child nodes.*

## 5.4.1.16 ShowGraphic

Method for the creation of a graphic view.

### SYNTAX

#### ShowGraphic (bstrObjectId, bstrSettings)

Parameters	Description
bstrObjectId	A string containing the ID of the object.
bstrSettings	Define the graphik settings. the meaning of the settings: [FLOOR:1] show floor [TRANS:0.5] transparency of floor is 0.5 (0 = totally transparent, 1 = opaque) [GRID:1] show grid [GRIDDIST:200] distance of grid lines is 200mm [GRIDCOL:255] grid color is red

### Script Return Value

None

### Example

#### Example

```
REM Manipulation der Position eines graphischen Objektes (nur Translation)
Sub main (id)
call Graphic.ShowGraphic(id,
"[FLOOR:1] [TRANS:0.5] [GRID:1] [GRIDDIST:200] [GRIDCOL:255] ")
End sub
```

## 5.5 Class ScriptItemDialog

The [Table 15](#) describes the class scripitem dialog methods.

**Table 15: Class ScriptItem Dialog Methods**

Method	Description
<a href="#">MessageBox</a>	Display a standard message box.
<a href="#">MessageBoxExt</a>	Display a message box (select symbol and buttons, e.g. Abort/Retry/Ignore).
<a href="#">CreateInputControl</a>	Predefine a control for an input box.
<a href="#">ModifyInputControl</a>	Modify an input box edit control (read only, length, and value check).
<a href="#">InputBox</a>	Display an input box.
<a href="#">GetInputControlValue</a>	Retrieve the current values of the controls.
<a href="#">GetValueCB</a>	Retrieve the current values of the OpenProject "ComboBox" controls.(script action context required)
<a href="#">GetValueCKB</a>	Retrieve the current values of the OpenProject "CheckBox" controls.(script action context required)
<a href="#">SetValueEdit</a>	Retrieve the current values of the OpenProject "Edit" controls.
<a href="#">SetValueCB</a>	Set values of the OpenProject "ComboBox" controls.(script action context required)
<a href="#">SetValueCKB</a>	Set values of the OpenProject "CheckBox" controls.(script action context required)
<a href="#">SetValueEdit</a>	Set values of the OpenProject "Edit" controls.(script action context required)
<a href="#">FileSelector (PE 5.8)</a>	Display a standard file selector.
<a href="#">FileSelector (PE 5.9)</a>	Display a standard file selector.
<a href="#">PropSetValues</a>	Write/append values to a control (script action context required).
<a href="#">PropGetValues</a>	Read the possible values from a control (script action context required).
<a href="#">PropGetCurrentValue</a>	Read the current value from a combo box, list box or radio button control (script action context required).
<a href="#">PropModifyAccess</a>	Change control's access.(script action context required)
<a href="#">PropModifyBGColor</a>	Change control's background color.(script action context required)
<a href="#">PropModifyFGColor</a>	Change control's foreground color.(script action context required)
<a href="#">PropModifyFontSize</a>	Change control's font size.(script action context required)
<a href="#">PropModifyFontStyle</a>	Change control's font style.(script action context required)
<a href="#">PropModifyFontType</a>	Change control's font type.(script action context required)
<a href="#">PropModifyRep</a>	Change control's representation.(script action context required)
<a href="#">PropModifyStyle</a>	Change control style (script action context required).
<a href="#">PropModifyVisibility</a>	Hide/Show customization layout objects
<a href="#">Redraw</a>	Redraw dialog.(script action context required)
<a href="#">RedrawAttribute</a>	Redraw attribute.(script action context required)
<a href="#">RedrawGroup</a>	Redraw group.(script action context required)
<a href="#">RedrawPage</a>	Redraw page.(script action context required)
<a href="#">ResetItem</a>	Reset the script item to its initial state.

## 5.5.1 List of XScriptItemDialog Methods

### 5.5.1.1 MessageBox



**SYNTAX****MessageBox( bstrCaption, bstrText)**

Parameters	Description
bstrCaption	Dialog caption.
bstrText	Displayed text.

**Script Return Value**

None

**Example****Example**

```

REM Displays a standard message box
REM entry parameter [id] is not used in this script
sub main(id)
    caption = "MessageBox"
    message = "Hello world!"
    call Dialog.MessageBox(caption, message)
end sub

```

**5.5.1.2 MessageBoxExt****SYNTAX****MessageBoxExt(bstrCaption, bstrText, bstrType)**

Parameters	Description
bstrCaption	Dialog caption.
bstrText	Displayed text.
bstrType	The type of the message box to be displayed (see Script Return Values).

**Script Return Value**

int\* iRetVal (depends on value of bstrType)

BstrType	Script Return Value	Description
ABORT_RETRY_IGNORE	3	ABORT
	4	RETRY
	5	IGNORE
ALL_ONE_NO	6	ALL
	10	ONE
	7	NO
ERROR	1	OK
OK_CANCEL	1	OK
	2	CANCEL
WARNING	1	OK
YES_NO	6	YES
	7	NO
YES_NO_CANCEL	6	YES
	7	NO
	2	CANCEL
OK	1	OK

**Example****Example**

```

REM Display different types of message boxes
REM entry parameter [id] is not used in this skript

```

```

sub main (id)
  DisplayMessage("ABORT_RETRY_IGNORE")
  DisplayMessage("ALL_ONE_NO")
  DisplayMessage("ERROR")
  DisplayMessage("OK_CANCEL")
  DisplayMessage("WARNING")
  DisplayMessage("YES_NO")
  DisplayMessage("YES_NO_CANCEL")
  DisplayMessage("OK")
end sub
sub DisplayMessage(howtodisplay)
  RetVal=Dialog.MessageBoxExt("Caption", "Message", howtodisplay)
  call Dialog.MessageBox("Result", "Return Value is: " & RetVal)
end sub

```

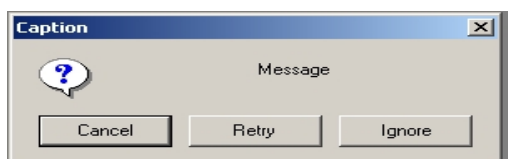


Figure 34: Abort/Retry/Cancel box

### 5.5.1.3 CreateInputControl

#### SYNTAX

**CreateInputControl(bstrCtrlId, bstrControlType, bstrAttribute, vDefaultValue)**

Parameters	Description
bstrCtrlId	A unique string identifier for the control.
bstrControlType	A valid specifier for the type of control.
bstrAttribute	Displayed prompt of the control.
vDefaultValue	Displayed default value of the control, or a list of values in case of "RadioButtons" and "ComboBox".

#### Script Return Value

None

Possible values for **bstrControlType**

Value	Description
EditString	Edit control containing strings.
EditStringNoLowerCase	Edit control containing strings containing no lower case characters.
EditInt	Edit control containing integers.
EditDouble	Edit control containing doubles.
MultiLineEdit	A multiline text control.
CheckBox	A check box. The value 0 means unchecked, whereas 1 means checked.
RadioButtons	A radio control. You must pass a list of prompts for the radio buttons. The buttons are enumerated internally beginning with 0.
ComboBox	A drop down list combo box. You must pass a list of pairs of values (visible item/internal item). The internal value is returned on OK.

**Example****Example**

```
'Displays an input box with the specified controls
'entry parameter [id] is not used in this script
sub main(id)
'Define Edit Control for InputBox (Parameter: control id, control type, prompt, default value)
    call Dialog.CreateInputControl("1", "EditString", "Your Name", "Peter")
...
'Define Combo Box Control (first value in array is default)
Dim combo(1,1)
combo(0,0) = "Visible Name 1"
combo(0,1) = "Internal Name 1"
combo(1,0) = "Visible Name 2"
combo(1,1) = "Internal Name 2"
call Dialog.CreateInputControl("6", "ComboBox", "Combo Selection", combo)
...
'Define Radio Button Control (first value in array is default)
Dim radio(2)
radio(0) = "A"
radio(1) = "B"
radio(2) = "C"
call Dialog.CreateInputControl("7", "RadioButtons", "Radio Selection", radio)
'display InputBox caption "Your Data"
ret = Dialog.InputBox("Your Data")
if ret=1 then
    'check content of edit controls by means of control id (see above)
    name=Dialog.GetInputControlValue("1")
    comboval=Dialog.GetInputControlValue("6")
    radioval=Dialog.GetInputControlValue("7")
    message="Name: " & name
else
    message="Dialog canceled."
end if
'Display message box
call Dialog.MessageBox("Check Input", message)
end sub
```

**Note**

*The buttons "Help" and "Default" displayed in the dialog are without function and therefore disabled.*



Figure 35: Example eines Dialog

### 5.5.1.4 ModifyInputControl

#### SYNTAX

**ModifyInputControl(bstrCtrlId, bstrCtrlTopic, vValue)**

Parameters	Description
bstrCtrlId	A unique string identifier for the control
bstrCtrlTopic	A valid specifier for the topic you like to control For example, "ReadOnly".
vValue	A valid value for the specified topic.

Possible values for **bstrCtrlTopic**

Value	Description	Validity
ReadOnly	The control is displayed read only if you pass "true".	Valid for all edit controls.
MaxChar	The length of the input is confined to MaxChar. Default is system dependent (32bit = 32767)	Valid for all edit controls.
MinLimit	The lower bound value of the input is confined to MinLimit. Default is system dependent (32bit = -2147483648)	Valid for all number edit controls.
MaxLimit	The lower bound value of the input is confined to MaxLimit. Default is system dependent (32bit = 2147483647)	Valid for all number edit controls.

#### Script Return Value

None

#### Example

#### Example

```
'Displays an input box with the specified controls
'entry parameter [id] is not used in this skript

sub main(id)

call Dialog.CreateInputControl("1", "EditString", "Name", "Peter")
call Dialog.ModifyInputControl("1", "ReadOnly", True)
...

call Dialog.CreateInputControl("2", "EditInt", "Number", "123")
call Dialog.ModifyInputControl("2", "MinLimit", 0)
call Dialog.ModifyInputControl("2", "MaxLimit", 1000)
...
```

```
call Dialog.CreateInputControl("3", "EditStringNoLowerCase",
"NLC", "ABCDEFGF")
call Dialog.ModifyInputControl("3", "MaxChar", 20)
end sub
```

### 5.5.1.5 InputBox

#### SYNTAX

#### CreateInputControl(bstrCaption)

Parameter	Description
bstrCaption	Caption of the file selection dialog.

#### Script Return Value

Parameters	Description
-1	If the object is derived from this class.
0	If the object is not derived from this class.



#### Note

*In VBScript -1 evaluates to True or vbTrue, but in JScript +1 evaluates to true.*

#### Example

#### Example

```
'Displays an input box with the specified controls
'entry Parameter [id] is not used in this skript
sub main(id)
'Define Edit Control for InputBox (Parameter: control id, control
type, prompt, default value)
call Dialog.CreateInputControl("1", "EditString", "Your Name",
"Peter")
...
'Define Combo Box Control (first value in array is default)
Dim combo(1,1)
combo(0,0) = "Visible Name 1"
combo(0,1) = "Internal Name 1"
combo(1,0) = "Visible Name 2"
combo(1,1) = "Internal Name 2"
call Dialog.CreateInputControl("6", "ComboBox", "Combo Selec-
tion", combo)
...
'Define Radio Button Control (first value in array is default)
Dim radio(2)
radio(0) = "A"
radio(1) = "B"
radio(2) = "C"
call Dialog.CreateInputControl("7", "RadioButtons", "Radio Se-
lection", radio)
'display InputBox caption "Your Data"
ret = Dialog.InputBox("Your Data")
if ret=1 then
'check content of edit controls by means of control id (see
above)
name=Dialog.GetInputControlValue("1")
comboval=Dialog.GetInputControlValue("6")
radioval=Dialog.GetInputControlValue("7")
message="Name: " & name
else
message=" Dialog canceled."
end if
'Display message box
call Dialog.MessageBox("Check Input", message)
end sub
```

### 5.5.1.6 GetInputControlValue

#### SYNTAX

**GetInputControlValue(bstrCtrlId)**

Parameters	Description
bstrCtrlId	The unique string identifier of the control

#### Script Return Value

Parameters	Description
Current Value of the control.	If dialog has been left by "OK".
Empty	If dialog has been left by "Cancel".

Return values dependent on **Controltyp** (Please refer to the [CreateInputControl](#))

EditString	A string.
EditInt	An integer.
EditDouble	A double value (floating point number).
MultiLineEdit	A string.
CheckBox	0 if checked, 1 if unchecked.
RadioButtons	The order number of the selected radio button (the order number of the first button is 0).
ComboBox	The internal value of the combo box (not the visible).

#### Example

#### Example GetInputControlValue

```
'Displays an input box with the specified controls
'entry parameter [id] is not used in this skript
sub main(id)
'Define Edit Control for InputBox (Parameter: control id, control type, prompt, default value)
call Dialog.CreateInputControl("1", "EditString", "Your Name", "Peter")
...
'Define Combo Box Control (first value in array is default)
Dim combo(1,1)
combo(0,0) = "Visible Name 1"
combo(0,1) = "Internal Name 1"
combo(1,0) = "Visible Name 2"
combo(1,1) = "Internal Name 2"
call Dialog.CreateInputControl("6", "ComboBox", "Combo Selection", combo)
...
'Define Radio Button Control (first value in array is default)
Dim radio(2)
radio(0) = "A"
radio(1) = "B"
radio(2) = "C"
call Dialog.CreateInputControl("7", "RadioButtons", "Radio Selection", radio)
'display InputBox caption "Your Data"
ret = Dialog.InputBox("Your Data")
if ret=1 then
'check content of edit controls by means of control id (see above)
name=Dialog.GetInputControlValue("1")
comboval=Dialog.GetInputControlValue("6")
radioval=Dialog.GetInputControlValue("7")
message="Name: " & name
else
```

```

        message="Dialog canceled."
    end if
'Display message box
call Dialog.MessageBox("Check Input", message)
end sub

```

### 5.5.1.7 GetValueCB

#### SYNTAX

**GetValueCB (int ctrlType; VARIANT\* varCBVals)**

Parameters	Description
ctrlType	An int identifier for the control
varCBVals	A list of values of the "ComboBox"

#### Script Return Value

If successful, an array of the Combobox entries.

Possible values for ctrlType

Value	Description
0	GLOBAL_FILTER
1	PRODUCT_FILTER
2	PROCESS_FILTER
3	RESOURCE_FILTER

### 5.5.1.8 GetValueCKB

#### SYNTAX

**GetValueCKB (int ctrlType; int iValue)**

Parameter	Description
ctrlType	An int identifier for the control

#### Script Return Value

If successful, an array of the Combobox entries

### 5.5.1.9 GetValueEdit

#### SYNTAX

**GetValueEdit (int ctrlType; BSTR\* bstrEditVal)**

Parameter	Description
ctrlType	An int identifier for the control

#### Script Return Value

If successful the Edit Controls entry.

### 5.5.1.10 SetValueCB

#### SYNTAX

**SetValueCB (int ctrlType; VARIANT\* pvarValues)**

Parameter	Description
ctrlType	An int identifier for the control
pvarValues	A list of values of the "ComboBox"

#### Script Return Value

None

Possible values for ctrlType

Value	Description
0	GLOBAL_FILTER
1	PRODUCT_FILTER
2	PROCESS_FILTER
3	RESOURCE_FILTER

#### 5.5.1.11 SetValueCKB

##### SYNTAX

**SetValueCKB (int ctrlType; int iCKBVal)**

Parameters	Description
ctrlType	An int identifier for the control
iCKBVal	"1" check, "0" uncheck

##### Script Return Value

None

#### 5.5.1.12 SetValueEdit

##### SYNTAX

**SetValueEdit (int ctrlType; BSTR bstrEditVal)**

Parameters	Description
ctrlType	An int identifier for the control
bstrEditVal	A string entry

##### Script Return Value

None

#### 5.5.1.13 FileSelector (PE 5.7)

##### SYNTAX

**FileSelector(bstrCaption, bstrDefaultFile, vFilterArray)**

Parameters	Description
bstrCaption	Caption of the file selection dialog
bstrDefaultFile	A valid default file including path information. You can use wildcards. If empty, the system "TEMP" directory is selected.
vFilterArray	An array of text/extension pairs used for filtering the displayed files, Please refer to the <a href="#">Example</a> . If the array is empty filtering is disabled. Otherwise the first entry (pair) is the default.

##### Script Return Value

BSTR bstrSelectedFile

Name (incl. Path) of the selected file	If successful and "Open" has been selected.
Empty	otherwise

##### Example

### Example

```
REM Displays a standard file selection dialog
REM entry parameter [id] is not used in this skript
sub main(id)
    default_filename="C:\temp\*.*"
    caption="Select a file ..."
    Dim filter(2,1)
    filter(0,0) = "Text Files (*.txt)"
    filter(0,1) = "*.txt"
```



```

filter(1,0) = "Comma Separated Files (*.csv)"
filter(1,1) = "*.csv"
filter(2,0) = "Logfiles (*.log)"
filter(2,1) = "*.log"
REM R6 Function Call
filename = Dialog.FileSelector(caption, default_filename,
filter)
REM R8 Function Call
'filename = Dialog.FileSelector("Open", caption, de-
fault_filename, filter)
if filename <> "" then
    MsgBox(filename)
else
    MsgBox("No file selected!")
end if
end sub

```

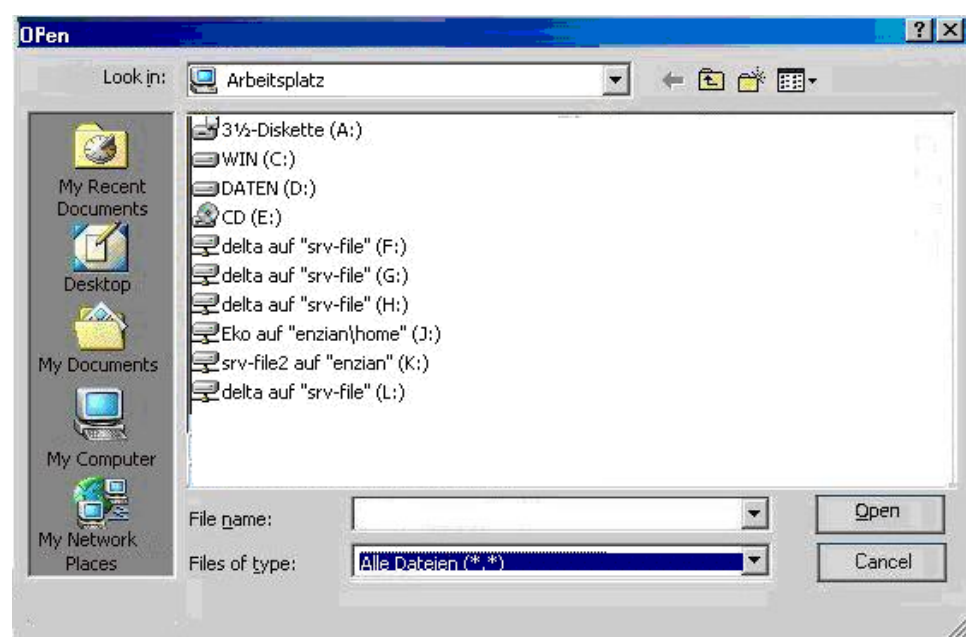


Figure 36: FileSelector PE 5.7

### 5.5.1.14 FileSelector (PE 5.9)

#### SYNTAX

**FileSelector(bstrMode, bstrCaption, bstrDefaultFile, vFilterArray)**

Parameters	Description
bstrMode	"Open" or "SaveAs" dependent on how you like to display the dialog button.
bstrCaption	Caption of the file selection dialog.
bstrDefaultFile	A valid default file including path information. You can use wildcards. If empty, the system "TEMP" directory is selected.
vFilterArray	An array of text/extension pairs used for filtering the displayed files, <i>Please refer to the <a href="#">Example</a>.</i>  If the array is empty filtering is disabled. Otherwise the first entry (pair) is the default.

#### Script Return Value

bstrSelectedFile

Parameters	Description
Name (incl. Path) of the selected	If successful and a file has been selected.

Parameters	Description
file	
Empty	otherwise

**Example****Example**

```

REM Displays a standard file selection dialog
REM entry parameter [id] is not used in this skript
sub main(id)
    default_filename="C:\temp\*.*"
    caption="Select a file..."
    Dim filter(2,1)
    filter(0,0) = "Text Files (*.txt)"
    filter(0,1) = "*.txt"
    filter(1,0) = "Comma Separated Files (*.csv)"
    filter(1,1) = "*.csv"
    filter(2,0) = "Logfiles (*.log)"
    filter(2,1) = "*.log"
    REM R6 Function Call
    'filename=Dialog.FileSelector(caption, default_filename, fil-
    ter)
    REM R8 Function Call
    filename = Dialog.FileSelector("Open", caption, de-
    fault_filename, filter)
    if filename <> "" then
        MsgBox(filename)
    else
        MsgBox("No file selected!")
    end if
end sub

```



Figure 37: FileSelector PE 5.9

**5.5.1.15 PropSetValues****SYNTAX**

**PropSetValues(bstrObjectId, bstrAttributeId, iLinkType, vVarArray)**

Parameters	Description
bstrObjectId	The ID of the object under consideration.
bstrAttributeId	Name (=unique key) of the attribute that is represented by the control.
iLinkType	0 = concat (append the values provided by the script to those read from the database)

Parameters	Description
	1 = use external (replace the controls original values by the external values provided by the script)
vVarArray	Array containing the new value(s). The array size and dimension depends on attribute and control type.  For combo boxes and list boxes, the first column of the array contains the visible, the second value the internal (database persistent) value.

**Script Return Value**

none

**Example****Example**

```

sub set_carbodyposition(object_id, classification)
' this example is executable for a standard process component
Dim Values
...
if classification= "c3" then ' set carbodyposition menu for core
processes
ReDim Values(2,1)
Values(0,0) = "to core process 1 (Script)"
Values(0,1) = "cp1"
Values(1,0) = "to core process 2 (Script)"
Values(1,1) = "cp2"
Values(2,0) = "to core process 3 (Script)"
Values(2,1) = "cp3"
use_external = 1
call Dialog.PropSetValues(object_id,"carbodyposition",
use_external, Values)
end if
...
end sub

```

**Note**

Due to the modal character of the properties dialog, this function can only be used in the context of the script actions [Init Properties](#) and [Change Properties](#).

**5.5.1.16 PropGetValues****SYNTAX****PropGetValues(bstrObjectId, bstrAttributId, iLinkType)**

Parameters	Description
bstrObjectId	The ID of the object under consideration.
bstrAttributId	Name (=unique key) of the attribute that is represented by the control.
iLinkType	0 = concat (read all values - those from the database plus the values previously provided by a script. 1 = use external (get only those values appended by a script - if any)

**Script Return Value**

An array of values containing the content the control. The size and dimension of the array depends on the control type. For edit controls (strings, numbers, dates), this is a single value while, for example, for combo boxes and list boxes this is a list of name/value pairs.

**Example****Example**

```

function sa_initproperties(object_ids)
' this example is executable for a standard process component
attribute_id = "classification"
text = "Attribute: " & attribute_id & vbCrLf & vbCrLf
allvalues = Dialog.PropGetValues(object_ids(0), attribute_id ,

```

```

0)
dimx = ubound(allvalues, 1)

for i = 0 to dimx
visible_value = allvalues(i,0)
internal_value = allvalues(i,1)
text = text & visible_value & "(" & internal_value & ")" &
vbCRLF
next

caption = "Value list"
call Dialog.MessageBox(caption, text)

CurrentValue = Dialog.PropGetCurrentValue(object_ids(0), "clas-
sification")

caption = "Current value"
text = "Attribute: " & attribute_id & vbCRLF & "Current value: "
& CurrentValue
call Dialog.MessageBox(caption, text)

call set_carbodyposition(object_ids(0), CurrentValue)
call set_nameshort(object_ids(0))
end function

```



### Note

*Due to the modal character of the properties dialog, this function can only be used in the context of the script actions [Init Properties](#) and [Change Properties](#).*

## 5.5.1.17 PropGetCurrentValue

### SYNTAX

**PropGetCurrentValue(bstrObjectId, bstrAttributeld)**

Parameters	Description
bstrObjectId	The ID of the object under consideration.
bstrAttributeld	Name (=unique key) of the attribute that is represented by the control.

### Script Return Value

Retrieves the current content of a control. This is required for combo box, list box, and radio button controls.

### Example

### Example

```

function sa_initproperties(object_ids)
' this example is executable for a standard process component
attribute_id = "classification"
text = "Attribute: " & attribute_id & vbCRLF & vbCRLF
allvalues = Dialog.PropGetValues(object_ids(0), attribute_id ,
0)
dimx = ubound(allvalues, 1)
for i = 0 to dimx
visible_value = allvalues(i,0)
internal_value = allvalues(i,1)
text = text & visible_value & "(" & internal_value & ")" &
vbCRLF
next
caption = "Value list"
call Dialog.MessageBox(caption, text)
CurrentValue = Dialog.PropGetCurrentValue(object_ids(0), "clas-
sification")
caption = "Current value"
text = "Attribute: " & attribute_id & vbCRLF & "Current value: "

```

```
& CurrentValue
call Dialog.MessageBox(caption, text)
call set_carbodyposition(object_ids(0), CurrentValue)
call set_nameshort(object_ids(0))
end function
```

**Note**

*Due to the modal character of the properties dialog, this function can only be used in the context of the script actions [Init Properties](#) and [Change Properties](#).*

**5.5.1.18 PropModifyAccess****SYNTAX**

**PropModifyAccess(bstrObjectId, layoutObjectType, layoutObjectId, iMode)**

Parameters	Description
bstrObjectId	The ID of the object under consideration.
layoutObjectType	The type of the object under consideration.
bstrAttributeId	Name (=unique key) of the attribute that is represented by the control.
iMode	Accessmode: "1" read only and "0" writable.

**Script Return Value**

None

**Example****Example**

```
function sa_initproperties(object_ids)
oid = data.GetAttributebyId (object_ids(0), "oid")

attribute_id = "nameshort"
'Sets "nameshort" to "read only"
call Dialog.PropModifyAccess(oid, "attribute", attribute_id, 1)
end function
```

**5.5.1.19 PropModifyBGColor****SYNTAX**

**PropModifyBGColor(bstrObjectId, layoutObjectType, layoutObjectId, bstrColor)**

Parameters	Description
bstrObjectId	The ID of the object under consideration.
bstrAttributeId	Name (=unique key) of the attribute that is represented by the control.
iControlBGColor	Color e.g. "#FFFFFF" (white), "#000000" (black) etc.

**Script Return Value**

None

**Example****Example**

```
function sa_initproperties(object_ids)
oid = data.GetAttributebyId (object_ids(0), "oid")

attribute_id = "nameshort"
'set BGColor "nameshort" to blue
call Dialog.PropModifyBGColor(oid, "attribute", attribute_id, "#0000FF")
end function
```

**5.5.1.20 PropModifyFGColor****SYNTAX**

**PropModifyFGColor(bstrObjectId, layoutObjectType, layoutObjectId, bstrColor)**

Parameter	Description
bstrObjectId	The ID of the object under consideration.
bstrAttributeld	Name (=unique key) of the attribute that is represented by the control.
iControlFGColor	Color e.g. "#FFFFFF" (white), "#000000" (black) etc.

**Script Return Value**

None

**Example****Example**

```
function sa_initproperties(object_ids)
oid = data.GetAttributebyId (object_ids(0), "oid")

attribute_id = "nameshort"
'set FGColor "nameshort" to blue
call Dialog.PropModifyFGColor(oid, "attribute", attribute_id, "#0000FF")

end function
```

**5.5.1.21 PropModifyFontSize****SYNTAX****PropModifyFontSize(bstrObjectId, layoutObjectType, layoutObjectId, bstrSize)**

Parameters	Description
bstrObjectId	The ID of the object under consideration.
bstrAttributeld	Name (=unique key) of the attribute that is represented by the control.
iControlStyle	This attribute represents a bit mask. Currently only the values 0 (default) and 1 (control will be set to "read only") are supported.

**Script Return Value**

None

**Example****Example**

```
function sa_initproperties(object_ids)
oid = data.GetAttributebyId (object_ids(0), "oid")

attribute_id = "nameshort"
'set attribute "nameshort" to fontsize 20
call Dialog.PropModifyFontSize(oid, "attribute", attribute_id, "20")

end function
```

**5.5.1.22 PropModifyFontStyle****SYNTAX****PropModifyFontStyle(bstrObjectId, layoutObjectType, layoutObjectId, bstrStyle)**

Parameters	Description
bstrObjectId	The ID of the object under consideration.
layoutObjectType	The type of the object under consideration.
bstrAttributeld	Name (=unique key) of the attribute that is represented by the control.
bstrStyle	FontStyle: "B" bold, "I" italic, "U" underline or combinations e.g.: "B U I".

**Script Return Value**

None

**Example**

**Example**

```
function sa_initproperties(object_ids)
oid = data.GetAttributebyId (object_ids(0), "oid")
attribute_id = "nameshort"
'set nameshort to "underline" style
call Dialog.PropModifyFontStyle(oid, "attribute", attribute_id, "u")
end function
```

**5.5.1.23 PropModifyFontType****SYNTAX**

**PropModifyFontType(bstrObjectId, layoutObjectType, layoutObjectId, bstrStyle)**

Parameters	Description
bstrObjectId	The ID of the object under consideration.
layoutObjectType	The type of the object under consideration.
bstrAttributeld	Name (=unique key) of the attribute that is represented by the control.
bstrType	FontType, For example, "MS Sans Serif" etc.

**Script Return Value**

None

**Example****Example**

```
function sa_initproperties(object_ids)
oid = data.GetAttributebyId (object_ids(0), "oid")
attribute_id = "nameshort"
'set attribute "nameshort" to type "Arial"
call Dialog.PropModifyFontType(oid, "attribute", attribute_id, "Arial")
end function
```

**5.5.1.24 PropModifyRep****SYNTAX**

**PropModifyRep(bstrObjectId, layoutObjectType, layoutObjectId, bstrStyle)**

Parameters	Description
bstrObjectId	The ID of the object under consideration.
layoutObjectType	The type of the object under consideration.
bstrAttributeld	Name (=unique key) of the attribute that is represented by the control.
bstrStyle	Representation, a combination of color and font modifications at once: For example, "<bgcolor>#F00000</bgcolor><fgcolor>FF0000<fontsize>20</fontsize><fontstyle>u</fontstyle><fonttype>Arial</fonttype>"

**Script Return Value**

None

**Example****Example**

```
function sa_initproperties(object_ids)
attribute_id = "nameshort"
'set representation of attribute "nameshort"
call Dialog.PropModifyRep(object_ids(0), "attribute", attribute_id,
"<bgcolor>#F00000</bgcolor><fgcolor>FF0000<fontsize>20</fontsize><fontstyle>u</fontstyle><fonttype>Arial</fonttype>")
end function
```

**5.5.1.25 PropModifyStyle**

**SYNTAX****PropModifyStyle(bstrObjectId, bstrAttributeld, iControlStyle)**

Parameters	Description
bstrObjectId	The ID of the object under consideration.
bstrAttributeld	Name (= unique key) of the attribute that is represented by the control.
iControlStyle	This attribute represents a bit mask. Currently only the values: 0 (default) 1 (control will be set to "read only") are supported.

**Script Return Value**

none

**Example****Example**

```
function sa_initproperties(object_ids)

attribute_id = "nameshort"
'set attribute "nameshort" to read only
call Dialog.PropModifyStyle(object_ids(0), attribute_id, 1)

end function
```

**Note**

*Due to the modal character of the properties dialog, this function can only be used in the context of the script actions [Init Properties](#) and [Change Properties](#).*

**5.5.1.26 PropModifyVisibility****SYNTAX****PropModifyStyle(bstrObjectId, layoutObjectType, layoutObjectId, iMode)**

Parameters	Description
bstrObjectId	The ID of the object under consideration.
layoutObjectType	The type of the object under consideration.
bstrAttributeld	Name (=unique key) of the attribute that is represented by the control.
iMode	Visibilitymode: "0" Show and "1" Hide.

**Script Return Value**

none

**Example****Example**

```
function sa_initproperties(object_ids)
oid = data.GetAttributebyId (object_ids(0), "oid")
attribute_id = "nameshort"

'Hides nameshort
call Dialog.PropModifyVisibility(oid, "attribute", attribute_id, 0)
end function
```

**5.5.1.27 Redraw****SYNTAX****Redraw(bstrObjectId)**

Parameter	Description
bstrObjectId	The ID of the object under consideration

**Script Return Value**

None

**Example**



**Example**

```
function sa_initproperties(object_ids)
    oid = data.GetAttributebyId (object_ids(0), "oid")
    attribute_id = "nameshort"
    'set attribute "nameshort" to read only
    call Dialog.PropModifyStyle(oid, attribute_id, 1)
    call Dialog.Redraw(oid)
end function
```

**5.5.1.28 RedrawAttribute****SYNTAX****RedrawAttribute(bstrObjectId, attributeName)**

Parameters	Description
bstrObjectId	The ID of the object under consideration.
attributeName	Name (=unique key) of the attribute that is represented by the control.

**Script Return Value**

None

**Example****Example**

```
function sa_initproperties(object_ids)
    oid = data.GetAttributebyId (object_ids(0), "oid")
    attribute_id = "nameshort"
    'set attribute "nameshort" to read only
    call Dialog.PropModifyStyle(oid, attribute_id, 1)

    call Dialog.RedrawAttribute(oid, "nameshort")
end function
```

**5.5.1.29 RedrawGroup****SYNTAX****RedrawGroup(bstrObjectId; groupId)**

Parameters	Description
bstrObjectId	The ID of the object under consideration.
groupId	Name (=unique key) of the group that is represented by the control

**Script Return Value**

None

**Example****Example**

```
function sa_initproperties(object_ids)
    oid = data.GetAttributebyId (object_ids(0), "oid")
    attribute_id = "nameshort"
    'set attribute "nameshort" to read only
    call Dialog.PropModifyStyle(oid, attribute_id, 1)
    call Dialog.RedrawGroup(oid, "1100")

end function
```

**5.5.1.30 RedrawPage****SYNTAX****RedrawPage(bstrObjectId; pageId);**

Parameters	Description
bstrObjectId	The ID of the object under consideration.
pageId	Number (=unique key) of the page.

**Script Return Value**

None

**Example****Example**

```
function sa_initproperties(object_ids)
oid = data.GetAttributebyId (object_ids(0), "oid")
attribute_id = "nameshort"
'set attribute "nameshort" to read only
call Dialog.PropModifyStyle(oid, attribute_id, 1)
call Dialog.RedrawPage(oid, "1000")
end function
```

## 5.6 Class ScriptItemList

The [Table 16](#) describes the class scriptitem list methods.

**Table 16: Class ScriptItem List Methods**

Method	Description
<a href="#">CreateListView(PE 5.8)</a>	Create a browser list view containing the passed objects. The list view serves as drag & drop source.
<a href="#">CreateListView (PE 5.9)</a>	Create a browser list view containing the passed objects. The list view serves as drag & drop source.
<a href="#">OpenFinder</a>	Launch project finder with a predefined context.
<a href="#">ResetItem</a>	Reset the script item to its initial state.

### 5.6.1 List of XscriptItemList Methods

#### 5.6.1.1 CreateListView(PE 5.8)

**SYNTAX**

**CreateListView(bstrParentObjectId, bstrEmpty1, bstrEmpty2, vChildArray)**

Parameters	Description
bstrParentObjectId	A string containing the ID of the parent object.
bstrEmpty1	Not used.
bstrEmpty2	Not used.
vChildArray	An array containing the object IDs of the objects to be displayed in the list.

**Script Return Value**

None

**Example****Example**

```
' Displays a list of objects in a browser list view
' The list view supports drag and drop of the objects,
' Use this z. B. to display unassigned objects and drag
' them to the components they should be related to.
sub main(id)
i=0
Dim child_ids()
child_id = Data.GetFirstChild(id, "nodes")
Do while child_id <> ""
ReDim Preserve child_ids(i)
base_child_id=Data.GetAttributebyId(child_id, "ergocompbase")
child_ids(i)=base_child_id
child_id = Data.GetNextChild(id, "nodes")
```

```

    i=i+1
  Loop
  number_of_children=i
  MsgBox("Number of children = " & cstr(number_of_children))
  if number_of_children > 0 then
    'R6 function call
    'call List.CreateListView(id, "Test", "Test", child_ids)
    'R8 function call
    call List.CreateListView(id, "List of children", child_ids)
  end if
end sub

```



### Note

*The drag and drop behavior depends on the passed objects. Generally it is recommended to avoid passing of subcompitem or relationship objects. Pass objects of a basic object type (For Example, ergocompbase).*

#### 5.6.1.2 CreateListView(PE 5.9)

##### SYNTAX

**CreateListView(bstrParentObjectId, bstrCaption, vChildArray)**

Parameters	Description
bstrParentObjectId	A string containing the ID of the parent object.
bstrCaption	The caption of the list view window.
vChildArray	An array containing the object IDs of the objects to be displayed in the list.

##### Script Return Value

None

##### Example

### Example

```

' Displays a list of objects in a browser list view
' The list view supports drag and drop of the objects,
' Use this e.g. to display unassigned objects and drag
' them to the components they should be related to.
sub main(id)
  parent_base_id = GetBase(id)
  i=0
  Dim child_ids()
  child_id = Data.GetFirstChild(parent_base_id, "nodes")
  Do while child_id <> ""
    ReDim Preserve child_ids(i)
    child_ids(i)=Data.GetAttributebyId(child_id, "ergocompbase")
    child_id = Data.GetNextChild(parent_base_id, "nodes")
    i=i+1
  Loop
  number_of_children=i
  MsgBox("Number of children = " & cstr(number_of_children))
  if number_of_children > 0 then
    'R6 function call
    'call List.CreateListView(id, "Test", "Test", child_ids)
    'R8 function call
    call List.CreateListView(id, "List of children",
child_ids)
  end if
end sub
function GetBase(id)
  GetBase = Data.GetAttributebyId(id, "ergocompbase")
end function

```



### Note

The drag and drop behavior depends on the passed objects. Generally, it is recommended to avoid passing of subcompitem or relationship objects. Pass objects of a basic object type (For example, *ergocompbase*).

#### 5.6.1.3 OpenFinder

##### SYNTAX

**OpenFinder(bstrProjectId, bstrTypeName, vSearchCriteria)**

Parameters	Description
bstrProjectId	The ID of the project for which the finder is to be launched.
bstrTypeName	The name of the type to be preselected.
vSearchCriteria	An array containing additional search criteria to be preselected.

##### Script Return Value

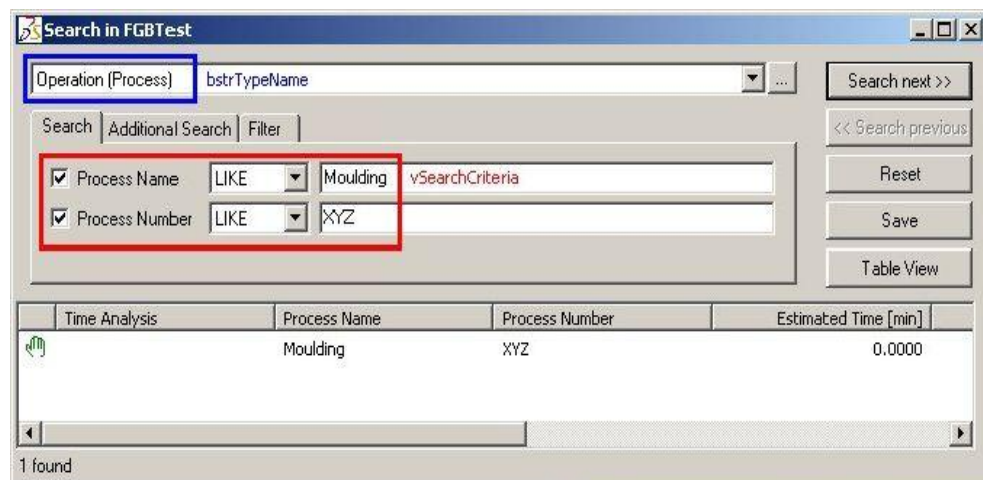
None

##### Example

#### Example

```
sub main(project_id)
  Dim searchAttributes
  mytypename = "Operation"
  ReDim searchAttributes(0, 1)
  searchAttributes(0,0) = "name"
  searchAttributes(0,1) = "Moulding"

  call List.OpenFinder(project_id, mytypename, searchAttributes)
end sub
```



### Note

Launching the project finder with a predefined context via script does not currently increase the finder performance. This is because it is independent of the predefinition; the entire configuration information is retrieved, before actually displaying the finder window.

## 5.7 Class ScriptItemQuery

The [Table 17](#) describes the class scriptitem query methods.

**Table 17: Class ScriptItem Query Methods**

Method	Description
<a href="#">SetQuery</a>	Define a (part of a) query.
<a href="#">SetConcatenator</a>	Set the logic concatenation of two subsequent queries (AND, OR, Brackets, ...).
<a href="#">GetFirstResult/GetNextResult</a>	Execute a query and retrieve the first result.
<a href="#">GetNextResult</a>	Retrieve the next result of a query (after a call to GetFirstResult).
<a href="#">GetResultCount</a>	Retrieve the number of results of a query.
<a href="#">GetOnlyFirstResult</a>	Execute a query and retrieve only the first result (enhances performance).
<a href="#">ResetSearch</a>	Reset the current query(ies) (if you have more than one query in a single script).
<a href="#">SetOrderAttribute</a>	Order the hits of a query by a predefined attribute.
<a href="#">BeginSubQuery</a>	Define the beginning of a subquery.
<a href="#">EndSubQuery</a>	Define the beginning of a subquery.
<a href="#">SetSubQuery</a>	Define a subquery.
<a href="#">SetQueryMode</a>	Sets the mode of the query (applies only when subqueries are used).
<a href="#">CacheResultIds</a>	Define if query results should be cached or not.
<a href="#">SetFetchingSize</a>	Select the number of results to fetch in a single server access.
<a href="#">SetOQLQuery</a>	Set directly an OQL Query. <b>This function is not supported for PE5.15 and later any longer.</b>
<a href="#">IsObjectValid</a>	Check if an object is valid with respect to the current filter settings.
<a href="#">UseProjectFilter</a>	Enable usage of project filter for the subsequent query.
<a href="#">IsFilterActive</a>	Check if any project filter is active.
<a href="#">ResetItem</a>	Reset the script item to its initial state.

## 5.7.1 List of ScriptItemQuery Methods

### 5.7.1.1 SetQuery

#### SYNTAX

**SetQuery(bstrSearchTable, bstrName, bstrOperator, varCompareTo)**

Parameters	Description
bstrSearchTable	The (object) table on which the query to perform (z. B. "ergocomplantdefault").
bstrName	The attribute on which the condition should be applied.
bstrOperator	The comparison operator (e.g. "=" (equals)).
varCompareTo	The value to which the current value of bstrName is to compare.

Possible values for **bstrOperator**

Value	Description
"="	Current value of bstrName equals varCompareTo.
"<>"	Current value of bstrName does not equal varCompareTo.
"<"	Current value of bstrName is less than varCompareTo.

Value	Description
"<="	Current value of bstrName is less than or equal to varCompareTo.
">"	Current value of bstrName is greater than varCompareTo.
">="	Current value of bstrName is greater than or equal to varCompareTo.
"NO"	Do not compare to any attribute (returns all objects of bstrSearchTable.

**Script Return Value**

None

**Example****Example**

REM When using the defaults you will find all product components of current project.  
 REM Then the skript must be executed on the current project node  
 REM since the passed id is used in the SetQuery statement

```

sub main (project_id)
  def_tablename= "ergocompproductdefault"
  def_attribute= "ergoproject"
  def_operator= "="
  def_vartocompare = project_id

  call Dialog.CreateInputControl("1", "EditString", "Objects (tablename) to search:", def_tablename)
  call Dialog.CreateInputControl("2", "EditString", "Attribute to compare:", def_attribute)
  call Dialog.CreateInputControl("3", "EditString", "Operator:", def_operator)
  call Dialog.CreateInputControl("4", "EditString", "Compare to value:", def_vartocompare )
  retval = Dialog.InputBox("Query Parameters")
  if retval <> False then

    tablename=Dialog.GetInputControlValue("1")
    attribute=Dialog.GetInputControlValue("2")
    operator=Dialog.GetInputControlValue("3")
    vartocompare=Dialog.GetInputControlValue("4")

    Query.ResetSearch
    call Query.SetQuery(tablename, attribute, operator, vartocompare)
    number_of_results = Query.GetResultCount
    MsgBox ("Number of results: " & number_of_results)
    if number_of_results > 0 then
      retval = MsgBox("Display the results?", 36)
      if retval = 6 then
        result_id=Query.GetFirstResult
        if result_id <> "" then
          Do while result_id <> ""
            value = Data.GetAttributeById (result_id,
"name")
            MsgBox(value)
            result_id=Query.GetNextResult
          Loop
        else
          MsgBox("No results.")
        end if
      end if
    end if
  end if
end if

```



```
end sub
```

### Note

- ▶ *bstrSearchTable* can be omitted in subsequent calls to *SetQuery* if the table name does not change (R6 and later only.).
- ▶ Use wildcards ("\*") for "like" queries, e.g. *bstrName* = "AB\*".
- ▶ You **can** pass configured names and physical names of search tables and search attributes.
- ▶ For example, pass *XDOScript* instead of *script* and *m\_name* instead of *name*. However, if you pass a real search table name you also **must** use real attribute names.
- ▶ If there is no configured name for a type or attribute you must pass the member variable string, For example "ergoproject".
- ▶ The case sensitivity extension "|CS" is available for PE5.13SP3 and later.
- ▶ The attribute extensions for user defined and/or date attributes are available for PE5.13SP3 and later.

## 5.7.1.2 SetConcatenator

### SYNTAX

#### SetConcatenator(bstrConcatenator)

Parameter	Description
BstrConcatenator	A string that describes the concatenation of subsequent queries.

### Script Return Value

None

Possible values for **bstrConcatenator**

Value	Description
"AND"	Query returns objects for which conditions A and B are true.
"OR"	Query returns objects for which conditions A or B are true.
"("	Open Bracket. Needed in nested conditions.
")"	Close Bracket. Needed in nested conditions.
"XOR"	Exclusive or Only valid if query mode SetQueryMode is "CompareSQ".
"IN_PRECEDING_QUERY"	Only the results of the preceding query are used in the subsequent condition. Only valid if query mode SetQueryMode is "CompareSQ".

### Example

#### Example

```
REM Advanced Query demonstrating usage of SetConcatenator
sub main (project_id)

typearea="ergocomplantdefault"
...
call Query.SetOrderAttribute(typearea, "name", "DESC")

call Query.SetConcatenator("(")
call Query.SetQuery (typearea, "name", "=", "*Drill*")
call Query.SetConcatenator("OR")
call Query.SetQuery (typearea, "nameshort", "=", "*2*")
call Query.SetConcatenator(")|AND")
```

```
call Query.SetQuery (typearea, "m_pErgoProject", "=",
project_id)

...
end sub
```



### Note

Use the separator "|" (pipe character) when combining concatenators. Please refer to the [Example](#).

#### 5.7.1.3 GetFirstResult

##### SYNTAX

##### GetFirstResult()

##### Script Return Value

If successful, the object id of the first result, or an empty string if there are no results to fetch.

##### Parameters

None

### Example

#### Example

```
REM When using the defaults you will find all product components
of current project.
REM Then the skript must be executed on the current project node
REM since the passed id is used in the SetQuery statement
sub main (project_id)
def_tablename= "ergocompproductdefault"
def_attribute= "m_pErgoProject"
def_operator= "="
def_vartocompare = project_id
call Dialog.CreateInputControl("1", "EditString", "Objects (tab-
lename) to search:", def_tablename)
call Dialog.CreateInputControl("2", "EditString", "Attribute to
compare:", def_attribute)
call Dialog.CreateInputControl("3", "EditString", "Operator:",
def_operator)
call Dialog.CreateInputControl("4", "EditString", "Compare to
value:", def_vartocompare )
retval = Dialog.InputBox("Query Parameters")
if retval <> False then
    tablename=Dialog.GetInputControlValue("1")
    attribute=Dialog.GetInputControlValue("2")
    operator=Dialog.GetInputControlValue("3")
    vartocompare=Dialog.GetInputControlValue("4")
    Query.ResetSearch
    call Query.SetQuery(tablename, attribute, "=", vartocompare)
    number_of_results = Query.GetResultCount
    MsgBox ("Number of results: " & number_of_results)
    if number_of_results > 0 then
        retval = MsgBox("Display the results?", 36)
        if retval = 6 then
            result_id=Query.GetFirstResult
            if result_id <> "" then
                Do while result_id <> ""
                    value = Data.GetAttributeById (result_id,
"name")
                    MsgBox (value)
                    result_id=Query.GetNextResult
                Loop
            else
                MsgBox("No results.")
            end if
        end if
    end if
end if
end sub
```



```

        end if
    end if
end if
end if
end sub

```

#### 5.7.1.4 GetNextResult

##### SYNTAX

##### GetNextResult()

##### Script Return Value

If successful, the object id of the next result, or an empty string if there are no more results to fetch.

#### Example

##### Example

```

REM When using the defaults you will find all product components
of current project.
REM Then the skript must be executed on the current project node
REM since the passed id is used in the SetQuery statement
sub main (project_id)
def_tablename= "ergocompproductdefault"
def_attribute= "m_pErgoProject"
def_operator= "="
def_vartocompare = project_id
call Dialog.CreateInputControl("1", "EditString", "Objects (tab-
lename) to search:", def_tablename)
call Dialog.CreateInputControl("2", "EditString", "Attribute to
compare:", def_attribute)
call Dialog.CreateInputControl("3", "EditString", "Operator:",
def_operator)
call Dialog.CreateInputControl("4", "EditString", "Compare to
value:", def_vartocompare )
retval = Dialog.InputBox("Query Parameters")
if retval <> False then
    tablename=Dialog.GetInputControlValue("1")
    attribute=Dialog.GetInputControlValue("2")
    operator=Dialog.GetInputControlValue("3")
    vartocompare=Dialog.GetInputControlValue("4")
    Query.ResetSearch
    call Query.SetQuery(tablename, attribute, "=", vartocompare)
    number_of_results = Query.GetResultCount
    MsgBox ("Number of results: " & number_of_results)
    if number_of_results > 0 then
        retval = MsgBox("Display the results?", 36)
        if retval = 6 then
            result_id=Query.GetFirstResult
            if result_id <> "" then
                Do while result_id <> ""
                    value = Data.GetAttributeById (result_id, "name")
                    MsgBox(value)
                    result_id=Query.GetNextResult
                Loop
            else
                MsgBox("No results.")
            end if
        end if
    end if
end if
end if
end sub

```



#### Note

Function must preceeded by a call to [GetFirstResult](#). The current position of the next child is stored in a pointer variable. A reverse iteration is not possible yet.

### 5.7.1.5 GetResultCount

#### SYNTAX

#### GetResultCount()

#### Script Return Value

The number of results, or zero if there are no results in the list.

#### Example

#### Example

```
REM When using the defaults you will find all product components
of current project.
REM Then the skript must be executed on the current project node
REM since the passed id is used in the SetQuery statement
sub main (project_id)
    def_tablename= "ergocompproductdefault"
    def_attribute= "m_pErgoProject"
    def_operator= "="
    def_vartocompare = project_id
    call Dialog.CreateInputControl("1", "EditString", "Objects
(tablename) to search:", def_tablename)
    call Dialog.CreateInputControl("2", "EditString", "Attribute
to compare:", def_attribute)
    call Dialog.CreateInputControl("3", "EditString", "Opera-
tor:", def_operator)
    call Dialog.CreateInputControl("4", "EditString", "Compare to
value:", def_vartocompare )
    retval = Dialog.InputBox("Query Parameters")
    if retval <> False then
        tablename=Dialog.GetInputControlValue("1")
        attribute=Dialog.GetInputControlValue("2")
        operator=Dialog.GetInputControlValue("3")
        vartocompare=Dialog.GetInputControlValue("4")
        Query.ResetSearch
        call Query.SetQuery(tablename, attribute, "=", vartocom-
pare)
        number_of_results = Query.GetResultCount
        MsgBox ("Number of results: " & number_of_results)
        if number_of_results > 0 then
            retval = MsgBox("Display the results?", 36)
            if retval = 6 then
                result_id=Query.GetFirstResult
                if result_id <> "" then
                    Do while result_id <> ""
                        value = Data.GetAttributebyId (result_id,
"name")
                        MsgBox(value)
                        result_id=Query.GetNextResult
                    Loop
                else
                    MsgBox("No results.")
                end if
            end if
        end if
    end if
end sub
```



#### Note

*You can check the result count before actually retrieving the results by [GetFirstResult](#).*

### 5.7.1.6 GetOnlyFirstResult

#### SYNTAX

#### GetOnlyFirstResult()

#### Script Return Value

If successful, the object id of the first result, or an empty string if there are no results to fetch.

#### Example

#### Example

```
sub main(id)
...
result_id=Query.GetOnlyFirstResult
...
end sub
```



#### Note

Similar to [GetFirstResult](#). However, the list of results returned from the query machine contains only one result.

A subsequent call to [GetNextResult](#) is not possible. Use this method to improve performance if you need only the first result of a (sorted) list.

### 5.7.1.7 ResetSearch

#### SYNTAX

#### ResetSearch()

#### Script Return Value

None

#### Example

#### Example

```
sub main(id)
...
call Query.ResetSearch
...
end sub
```



#### Note

This method must be called to prepare the query machine for a new query. All preceding [SetQuery](#) and [SetSubQuery](#) statements are cleared.

### 5.7.1.8 SetOrderAttribute

#### SYNTAX

#### SetOrderAttribute(bstrSearchTable, bstrAttributeName, bstrOrder)

Parameters	Description
bstrSearchTable	The (object) table on which the query to perform (For example, "ergocomplantdefault").
bstrAttributeName	The attribute by which the results should be ordered.
bstrOrder	Ascending or descending order.

Possible values for **bstrOrder**

Value	Description
"ASC"	Sort ascending.
"DESC"	Sort descending.

#### Script Return Value

None

#### Example

**Example**

```

REM Advanced Query demonstrating usage of SetConcatenator
sub main (project_id)
typearea="ergocomplantdefault"
...
call Query.SetOrderAttribute(typearea, "name", "DESC")

call Query.SetConcatenator("(")
call Query.SetQuery (typearea, "name", "=", "*Drill*")
call Query.SetConcatenator("OR")
call Query.SetQuery (typearea, "nameshort", "=", "*2*")
call Query.SetConcatenator(")|AND")
call Query.SetQuery (typearea, "m_pErgoProject", "=",
project_id)
...
end sub

```

**Note**

*Make sure that on using the global sort or attribute the returned result list objects actually have or inherit this attribute. Otherwise on returning the IDs the script will return the error "Attribute is unknown". This is of particular importance whenever more than one query is implemented in a script.*

**5.7.1.9 BeginSubQuery****SYNTAX****BeginSubQuery(bstrQueryId)**

Parameter	Description
bstrQueryId	Any user defined unique string which identifies to the subquery.

**Script Return Value**

None

**Example****Example**

```

REM Using a subquery
REM 1. Find all process components in project "project_name"
REM 2. Extract from 1 the results having name = "search_name"
REM Script is independent of entry id
sub main (id)
project_name="Project1"
search_name="*Weld*"
call Query.ResetSearch
call Query.BeginSubQuery("Sub1")
call Query.SetQuery("ergoproject", "name", "=", project_name)
call Query.EndSubQuery("Sub1")
call Query.SetSubQuery("ergocompprocessdefault",
"m_pErgoProject", "=", "Sub1")
call Query.SetQuery ("", "name", "=", search_name)
number_of_results = Query.GetResultCount
MsgBox ("Number of results: " & number_of_results)
  if number_of_results > 0 then
    retval = MsgBox("Display the results?", 36)
    if retval = 6 then
      result_id=Query.GetFirstResult
      if result_id <> "" then
        Do while result_id <> ""
          value = Data.GetAttributeById (result_id, "name")
          MsgBox(value)
          result_id=Query.GetNextResult
        Loop
      else
        MsgBox("No results.")
      end if
    end if
  end if
end sub

```



```

        end if
    end if
end if
end sub

```

### Note

*Nesting of subqueries is not allowed.*

Hence, each *BeginSubQuery* statement must be followed by a *EndSubQuery* statement before another *BeginSubQuery* call.

## 5.7.1.10 EndSubQuery

### SYNTAX

**EndSubQuery(bstrQueryId)**

Parameter	Description
bstrQueryId	Any user defined unique string which identifies to the subquery.

### Script Return Value

None

### Example

#### Example

```

REM Using a subquery
REM 1. Find all process components in project "project_name"
REM 2. Extract from 1 the results having name = "search_name"
REM Script is independent of entry id

sub main (id)
    project_name="Project1"
    search_name="*Weld*"

    call Query.ResetSearch
    call Query.BeginSubQuery("Sub1")
    call Query.SetQuery("ergoproject", "name", "=", project_name)
    call Query.EndSubQuery("Sub1")

    call Query.SetSubQuery("ergocompprocessdefault",
        "m_pErgoProject", "=", "Sub1")
    call Query.SetQuery ("", "name", "=", search_name)

    number_of_results = Query.GetResultCount
    MsgBox ("Number of results: " & number_of_results)
    if number_of_results > 0 then
        retval = MsgBox("Display the results?", 36)
        if retval = 6 then
            result_id=Query.GetFirstResult
            if result_id <> "" then
                Do while result_id <> ""
                    value = Data.GetAttributebyId (result_id, "name")
                    MsgBox(value)
                    result_id=Query.GetNextResult
                Loop
            else
                MsgBox("No results.")
            end if
        end if
    end if
end if
end sub

```



### Note

*Nesting of subqueries is not allowed.*

Hence, each *BeginSubQuery* statement must be followed by a *EndSubQuery* statement before another *BeginSubQuery* call.

### 5.7.1.11 SetSubQuery

#### SYNTAX

**SetSubQuery( bstrSearchTable, bstrName, bstrOperator, bstrQueryId)**

Parameters	Description
bstrSearchTable	The (object) table on which the query to perform (z. B. "ergocomplantdefault").
bstrName	The attribute on which the condition should be applied.
bstrOperator	The comparison operator (e.g. "=" (equals)).
varCompareTo	The value to which the current value of bstrName is to compare.

Possible values for **bstrOperator**

Value	Description
"="	Current value of bstrName equals varCompareTo.
"<>"	Current value of bstrName does not equal varCompareTo.
"<"	Current value of bstrName is less than varCompareTo.
"<="	Current value of bstrName is less than or equal to varCompareTo.
">"	Current value of bstrName is greater than varCompareTo.
">="	Current value of bstrName is greater than or equal to varCompareTo.
"NO"	Do not compare to any attribute (returns all objects of bstrSearchTable).

#### Script Return Value

None

### Example

#### Example

```
REM Using a subquery
REM 1. Find all process components in project "project_name"
REM 2. Extract from 1 the results having name = "search_name"
REM Script is independent of entry id
sub main (id)
project_name="Project1"
search_name="*Weld*"
call Query.ResetSearch
call Query.BeginSubQuery("Sub1")
call Query.SetQuery("ergoproject", "name", "=", project_name)
call Query.EndSubQuery("Sub1")
call Query.SetSubQuery("ergocompprocessdefault",
"m_pErgoProject", "=", "Sub1")
call Query.SetQuery("", "name", "=", search_name)
number_of_results = Query.GetResultCount
MsgBox ("Number of results: " & number_of_results)
if number_of_results > 0 then
retval = MsgBox("Display the results?", 36)
if retval = 6 then
result_id=Query.GetFirstResult
if result_id <> "" then
Do while result_id <> ""
value = Data.GetAttributebyId (result_id, "name")
MsgBox(value)
result_id=Query.GetNextResult
Loop
```

```

        else
        MsgBox("No results.")
        end if
    end if
end if
end sub

```

**Note**

- *bstrSearchTable* can be omitted in subsequent calls to *SetQuery* if the table name does not change(R6).
- If there is no configured name for a type or attribute you can/must pass the member variable string, For example. "m\_pErgoProject".
- Use wildcards ("\*") for "like" queries, For example, *bstrName* = "AB\*".

**5.7.1.12 SetQueryMode****SYNTAX****SetQueryMode (bstrSearchMode)**

Parameter	Description
bstrSearchMode	A string defining the search mode. Default is "Normal".

Possible values for **bstrSearchMode**

Value	Description
"Normal"	Normal query mode (default). Use this in ordinary queries and if you compare ordinary queries to subqueries.
"CompareSQ"	Use this mode if you like to compare results of different subqueries.

**Script Return Value**

None

**Example****Example**

```

sub main(id)
...
call Query.SetQueryMode("CompareSQ")
...
end sub

```

**Note**

The query mode is a global parameter which defaults to "Normal". Typically it is not required to change the mode, except if you like to compare subqueries.

That the passed values are case sensitive.

**5.7.1.13 CacheResultIds****SYNTAX****CacheResultIds(bFlag)**

Parameter	Description
bFlag	A global flag which tells the host to cache the retrieved results in a list or not. Default is True.

**Script Return Value**

None

**Example****Example**

```

sub main(id)

```

```
...
call Query.CacheResultIds(False)
...
end sub
```

#### 5.7.1.14 SetFetchingSize

##### SYNTAX

**SetFetchingSize**(ulFetchingSize);

Parameter	Description
ulFetchingSize	The number of results to fetch in a single server access. Default is 25.

##### Script Return Value

None

##### Example

#### Example

```
sub main(id)
...
call Query.SetFetchingSize(10)
...
end sub
```



##### Note

*If the actual number of results is less than ulFetchingSize then - of course - all current results are fetched at once.*

#### 5.7.1.15 SetOQLQuery

This function is not supported for R15 and later any longer.

##### SYNTAX

**SetOQLQuery**(bstrOQLQuery);

Parameter	Description
bstrOQLQuery	The OQL Query as a text string.

##### Script Return Value

none

#### Example

##### Example

```
sub main(id_notused)
REM OQL
Call Query.SetQueryMode("OQL")
oqlquery="SELECT * FROM XDOErgoPlanTypeExtent type;"
Call Query.SetOQLQuery(oqlquery)
REM "Normal" Query
REM call Query.SetQuery("XDOErgoPlanType", "", "NO", "")
number_of_results = Query.GetResultCount
MsgBox ("Number of results: " & number_of_results)
if number_of_results > 0 then
if number_of_results > 0 then
retval = MsgBox("Display the results?", 36)
if retval = 6 then
result_id=Query.GetFirstResult
if result_id <> "" then
Do while result_id <> ""
value = Data.GetAttributeById (result_id, "name")
MsgBox(value)
result_id=Query.GetNextResult
Loop
else
MsgBox("No results.")
end if
end if
```



```

    end if
  end if
end sub

```

**Note**

*Requires to set the query mode to "OQL" using `SetQueryMode` before calling `SetOQLQuery`.*

**5.7.1.16 IsObjectValid****SYNTAX****IsObjectValid(bstrObjectId)**

Parameters	Description
bstrObjectId	The object ID.

**Script Return Value**

Parameters	Description
bFlag	<ul style="list-style-type: none"> <li>• 1 (True) if the object is valid concerning the current filter settings.</li> <li>• 0 (False) if the object is invalid (would be filtered in the browser) concerning the current filter settings.</li> </ul>

**Example****Example**

```

sub main (project_id)
...
compvalid = Query.IsObjectValid(result_id)
...
end sub

```

**5.7.1.17 UseProjectFilter****SYNTAX****UseProjectFilter(bstrProjectId)**

Parameter	Description
bstrProjectId	The project ID.

**Note**

*You must call this function before executing a filtered query (before [GetFirstResult](#) or [GetResultCount](#), respectively).*

**Script Return Value**

None

**Example****Example**

```

sub main (project_id)
...
if radioval = 1 then
all Query.UseProjectFilter(project_id)
end if
...
end sub

```

**5.7.1.18 IsFilterActive****SYNTAX****IsFilterActive(bstrProjectId)**

Parameter	Description
bstrProjectId	The project ID.

**Script Return Value**

Parameter	Description
bFlag	<ul style="list-style-type: none"> <li>• -1 (True) if any filter is active.</li> <li>• 0 (False) if no filter is active.</li> </ul>

**Example****Example**

```

sub main (project_id)
filteractive = Query.IsFilterActive(project_id)
Dim radio(1)
radio(0) = "Normal Query"
radio(1) = "Filtered Query"
call Dialog.CreateInputControl("1", "RadioButtons", "Query
Mode", radio)
if filteractive = True then
    call Dialog.CreateInputControl("2", "EditString", "Project
filter", "active")
else
    call Dialog.CreateInputControl("2", "EditString", "Project
filter", "inactive")
end if
call Dialog.ModifyInputControl("2", "ReadOnly", True)
retval = Dialog.InputBox("Query Parameters")
if retval <> False then
    tablename="ergocomplantdefault"
    attribute="m_pErgoProject"
    operator="="
    vartocompare=project_id
    radioval=Dialog.GetInputControlValue("1")
    Query.ResetSearch
    if radioval = 1 then
        call Query.UseProjectFilter(project_id)
    end if
    call Query.SetQuery(tablename, attribute, "=", vartocom-
pare)
    number_of_results = Query.GetResultCount
    MsgBox ("Number of results: " & number_of_results)
    if number_of_results > 0 then
        retval = MsgBox("Display the results?", 36)
        if retval = 6 then
            result_id=Query.GetFirstResult
            if result_id <> "" then
                Do while result_id <> ""
                    value = Data.GetAttributebyId (result_id, "name")
                    compvalid = Query.IsObjectValid(result_id)
                    if compvalid = True then
                        addstring = " valid"
                    else
                        addstring = " invalid"
                    end if
                    MsgBox(value + addstring)
                    result_id=Query.GetNextResult
                Loop
            else
                MsgBox("No results.")
            end if
        end if
    end if
end if
end sub

```

## 5.8 Class ScriptItemConfig

The [Table 18](#) describes the class scripitem config methods.

**Table 18: Class Scripitem Config Methods**

Method	Description
<a href="#">GetAllTypes</a>	Get all configuration types (incl. plantypes).
<a href="#">GetAllPlanTypes</a>	Get all plantypes of a distinct type.
<a href="#">GetTypeInfoTN</a>	Get type info by (unique) type name.
<a href="#">GetTypeInfo</a>	Get type info by type ID.
<a href="#">GetAllAttributes</a>	Get all attributes of a type or plantype.
<a href="#">GetAttributeInfoTN</a>	Get attribute info using type name and attribute name
<a href="#">GetAttributeInfo</a>	Get attribute info by attribute ID.
<a href="#">GetAttributeValueList</a>	Get attribute value list of an attribute.
<a href="#">GetAttributeValueInfo</a>	Get attribute value info by attribute value ID.
<a href="#">GetParentPCRelations</a>	Get all parent child relations for a distinct parent type (or plantype).
<a href="#">GetPCRelations</a>	Get all parent child relations between passed parent and child type.
<a href="#">GetPCRelationInfo</a>	Get PC relation info by PC relation ID.
<a href="#">ResetItem</a>	Reset the script item to its initial state.

### 5.8.1 List of XScriptItemConfig Methods

#### 5.8.1.1 GetAllTypes

##### SYNTAX

##### GetAllTypes()

##### Script Return Value

An array containing the unique IDs of all types and plantypes in the database.

##### Example

#### Example

```
sub main(notused)
types = Config.GetAllTypes
dimx = UBound(types, 1)
MsgBox ("Number of types: " + cstr(dimx+1))
for m = 0 To dimx
REM do something with types(m)
REM ... next
end sub
```



##### Note

For a PTS dependent list of plantypes see [GetAllPlanTypes](#).

#### 5.8.1.2 GetAllPlanTypes

##### SYNTAX

##### GetAllPlanTypes(bstrProjectId, bstrPlanTypeSetId, bstrTypeId)

Parameters	Description
bstrProjectId	A string containing the ID a project.
bstrPlanTypeSetId	A string containing the ID of a plantypeset.

Parameters	Description
bstrTypeid	A string containing the ID of a configuration type.

**Script Return Value**

An array containing the unique IDs of plantypes dependent on the in parameters. You must pass either the project ID (each project has one distinct plantypeset (PTS) ) or the PTS ID itself. You can optionally pass a type ID. If a type ID is passed, only the plantypes having this base type are returned.

**Example****Example**

```
sub main(project_id)
plantypes = Config.GetAllPlanTypes(project_id, "", "ergoplan-
type_plant")
dimx = UBound(plantypes, 1)
MsgBox ("Number of plantypes: " + cstr(dimx+1))
for m = 0 To dimx
REM do something with plantypes(m)
REM ... next
end sub
```

**Note**

For a list of all types and plantypes, Please refer to the [GetAllTypes](#).

**5.8.1.3 GetTypeInfoTN****SYNTAX**

**GetTypeInfoTN (bstrTypeName, bstrProperty)**

Parameters	Description
bstrTypeName	A string containing the unique typename of the type or plantype.
bstrProperty	A string containing name of the property.

**Script Return Value**

The current value of the requested property.

**Example****Example**

```
sub main(project_id)
uniquetypename = "ergocompproductdefault"
property = "captionsingular"
value = Config.GetTypeInfoTN(uniquetypename, property)
MsgBox(value)
end sub
```

**Note**

A unique typename is e.g. "ergocompproductdefault" or "script".  
For plantypes, the unique typename is a Global Unique Identifier, For example, "587f7aaf-5de0-417b-a6c9-64537fa31fc3" (For example, an "Operation").

**5.8.1.5 GetTypeInfo****SYNTAX**

**GetTypeInfo bstrTypeid, bstrProperty)**

Parameters	Description
bstrTypeid	A string containing the ID of the type pr plantype.
bstrProperty	A string containing name of the property.

**Script Return Value**

The current value of the requested property.

**Example****Example**

```

sub main(notused)
  Dim attributes(5)
  attributes(0) = "id"
  attributes(1) = "typename"
  attributes(2) = "realtypename"
  attributes(3) = "captionsingular"
  attributes(4) = "baseclasstypename"
  ...
  types = Config.GetAllPlanTypes(project_id, "", basetype)
  dimx = UBound(types, 1)
  For m = 0 To dimx
    For k = 0 To 4
      Value = Config.GetTypeInfo(types(m), attributes(k))
      .Cells(i, k + 1).Value = Value
    Next
    i = i + 1
  Next
  ...
end sub

```

**Note**

The type ID parameter of this function must contain the unique part of the POET object ID of the type/plantype to be examined, preceded by the string "cf", e.g "cf2436". This value is typically retrieved from a previous call to *GetAllPlanTypes* or *GetAllTypes* which return an array of POET object IDs.

**5.8.1.6 GetAllAttributes****SYNTAX****GetAllAttributes(bstrTypeName)**

Parameter	Description
bstrTypeName	A string containing the name of the configuration type or plantype (GUID).

**Script Return Value**

An array containing the unique IDs of all attributes.

**Example****Example**

```

sub main(id)
  ...
  attr = Config.GetAllAttributes("script")
  ...
end sub

```

**5.8.1.7 GetAttributeInfoTN****SYNTAX****GetAttributeInfoTN (BstrTypeName, bstrAttributeName, bstrProperty)**

Parameters	Description
bstrTypeName	A string containing the unique typename of the type or plantype.
bstrAttributeName	A string containing the unique name of the attribute.
bstrProperty	A string containing the name of the property.

**Script Return Value**

The current value of the requested property.

**Example****Example**

```
sub main(id)
```

```

defval = Config.GetAttributeInfoTN("script", "name", "defaultva-
lue")
MsgBox(defval)
end sub

```

### 5.8.1.8 GetAttributeInfo

#### SYNTAX

#### GetAttributeInfo (bstrAttributeID, bstrProperty)

Parameters	Description
bstrAttributeID	A string containing the unique ID of the attribute.
bstrProperty	A string containing the name of the property.

#### Script Return Value

The current value of the requested property.

#### Example

#### Example

```

sub main(id)
...
value = Config.value = Config.GetAttributeInfo(attr(m), "attri-
butename")
...
end sub

```



#### Note

The attribute IDs can be retrieved using a previous call to [GetAllAttributes](#).

### 5.8.1.9 GetAttributeValueList

#### SYNTAX

#### GetAttributeValueList (bstrTypeName, bstrAttributeName)

Parameters	Description
bstrTypeName	A string containing the name of the configuration type or plantype (GUID).
bstrAttributeName	A string containing the name of the attribute.

#### Script Return Value

An array containing the value list pairs of the requested attribute.

#### Example

#### Example

```

sub main(id)
attrlist = Config.GetAttributeValueList("script", "language-
type")
dimx = ubound(attrlist, 1)
for m=0 to dimx
if attrlist(m) <> "" then
value = Config.GetAttributeValueInfo(attrlist(m), "value")
MsgBox(value)
end if
next
end sub

```

### 5.8.1.10 GetAttributeValueInfo

#### SYNTAX

#### GetAttributeValueInfo (bstrAttributeValuelD, bstrProperty)

Parameters	Description
bstrAttributeValuelD	A string containing the id of the item from the attribute value list.
bstrProperty	A string containing the name of the property.

**Script Return Value**

The value of the property of the requested item from the attribute value list.

**Example****Example**

```
sub main(id)
attrlist = Config.GetAttributeValueList("script", "language-
type")
dimx = ubound(attrlist, 1)
for m=0 to dimx
  if attrlist(m) <> "" then
    value = Config.GetAttributeValueInfo(attrlist(m), "value")
    MsgBox(value)
  end if
next
end sub
```

**Note**

Typically follows a previous call to [GetAttributeValueList](#).

**5.8.1.11 GetParentPCRelations****SYNTAX**

**GetParentPCRelations(bstrParentTypeName, bstrConstraint)**

Parameters	Description
bstrParentTypeName	A string containing the unique name of the parent type.
bstrConstraint	Currently not used.

**Script Return Value**

An array containing the unique IDs of all PC relations for the given parent type.

**Example****Example**

```
sub main(id)
...
pcrels = Config.GetParentPCRelations(parent_type, constraint)
...
end sub
```

**Note**

The constraint parameter is currently not used.

**5.8.1.12 GetPCRelations****SYNTAX**

**GetPCRelations (bstrParentTypeName, bstrChildTypeName, bstrConstraint)**

Parameters	Description
bstrParentTypeName	A string containing the unique name of the parent type.
bstrChildTypeName	A string containing the unique name of the childtype.
bstrConstraint	Currently not used.

**Script Return Value**

An array containing the unique IDs of all PC relations for given parent type and child type.

**Example****Example**

```
sub main(id)
...
```

```
pcrels = Config.GetPCRelations(parent_type, child_type, constraint)
...
end sub
```

**Note**

*The constraint parameter is currently not used.*

**5.8.1.13 GetPCRelationInfo****SYNTAX**

**GetPCRelationInfo (bstrPCRelationId, bstrProperty);**

Parameters	Description
bstrPCRelationId	A string containing the unique ID of the PC relationship.
bstrChildTypeName	A string containing the name of the property to obtain.

**Script Return Value**

The current value of the property of the given PC relation.

**Example****Example**

```
sub main(id)
...
value = Config.GetPCRelationInfo(pcrels(m), "id")
...
end sub
```

**Note**

*Typically follows a previous call to [GetParentPCRelations](#) or [GetPCRelations](#).*

**5.8.1.14 ResetItem****SYNTAX**

**ResetItem()**

**Example****Example**

```
sub main
...
Config.ResetItem
...
end sub
```

**Script Return Value**

None

**Note**

*This method resets the script item back into its initial state. All static members are cleared and initialized to a default value. It is implemented for all XScriptItem classes but usage is only required if there are static members at all **and** you are going to reuse some of the members while running the script (For example, children lists, queries, ...) The method is implicitly called whenever you run a script.*

## 5.9 Class ScriptItemConvert

The [Table 19](#) describes class scriptitem convert methods.

**Table 19: Class ScriptItem Convert Methods**



Method	Description
<a href="#">Rtf2PlainText</a>	
<a href="#">TranslateText</a>	
<a href="#">ResetItem</a>	Reset the script item to its initial state.

## 5.9.1 List of XScriptItemConvert Methods

### 5.9.1.1 Rtf2PlainText

#### SYNTAX

**Rtf2PlainText(bstrRtf)**

Parameter	Description
bstrRtf	An RTF formatted text string.

#### Script Return Value

Parameter	Description
bstrPlainText	The plain text without RTF header, formatting and control sequences.

#### Example

#### Example

```
sub main(id)
    rtf_text = Data.GetAttributebyId(id, "note")
    rtf_plain = Convert.Rtf2PlainText(rtf_text)

    MsgBox("RTF formatted text" & vbCRLF & vbCRLF & rtf_text)
    MsgBox("Plain text" & vbCRLF & vbCRLF & rtf_plain)
end sub
```

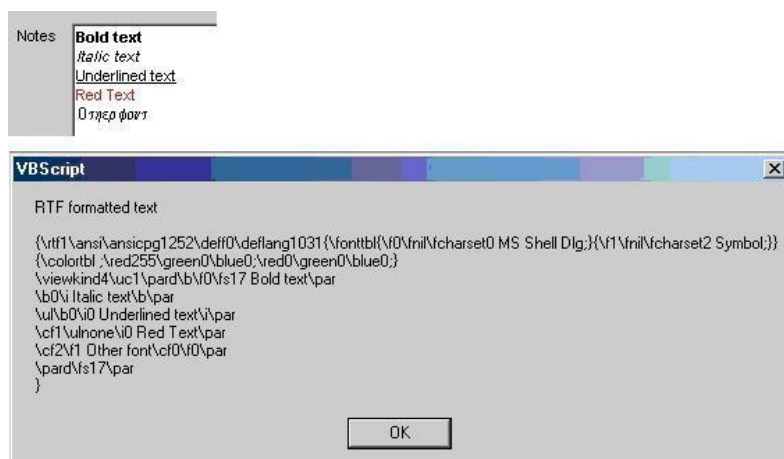


Figure 38: RTF Formatted Text

### 5.9.1.2 TranslateText

#### SYNTAX

**TranslateText(bstrText)**

Parameter	Description
bstrText	The text to be translated.

#### Script Return Value

Parameter	Description
bstrTextTranslated	The translated text

**Example****Example**

```
sub main(notused)

orgText = "This is the Process Engineer VBScript documentation."
translatedText = Convert.TranslateText(orgText)

MsgBox("Original Text: " & orgText & vbCRLF & "Translated Text: " & translatedText)

end sub
```

**Figure 39: VB Script**

When using the functions `MessageBox`, `MessageBoxExt` to display messages on the screen, `Convert.TranslateText` is **not** required. These dialog classes are using build-in translation functionalities.

## 5.10 Class ScriptItemVersion

The [Table 20](#) describes class scriptitem version methods.

**Table 20: Class ScriptItem Version Methods**

Method	Description
<a href="#">Create</a>	Create a new version.
<a href="#">CreateDeep</a>	Create new versions recursively.
<a href="#">CheckOut</a>	Check out (= Create and immediately use) this version.
<a href="#">CheckOutDeep</a>	Check out (= Create and immediately use) these versions recursively.
<a href="#">SetPlanningState</a>	Set the planning state of a version.
<a href="#">GetPlanningState</a>	Get the planning state of a version.
<a href="#">GetPlanningStates</a>	Get all possible planning states (global and local).
<a href="#">GetFirstVersion</a>	Get the first version of a component.
<a href="#">CreateReleaseTable</a>	Creates a release table.
<a href="#">GetReleaseTable</a>	Allocating an existing release table.
<a href="#">CreateSFI</a>	Create new version
<a href="#">CreateDeepSFI</a>	Create new versions recursively.
<a href="#">CheckOutSFI</a>	Check out this version.
<a href="#">CheckOutDeepSFI</a>	Check out these versions recursively
<a href="#">PDXMLFileCreated</a>	If the extended effectivity is part of this versions release table, the method <code>PDXMLFileCreated</code> enters the creation of a PDXMLFile to it.

Method	Description
<a href="#">Promote</a>	Promotes a component to a higher planning state.
<a href="#">CanPlanningStateBeModified</a>	Whether the modification of component's planning state is allowed.
<a href="#">Use</a>	Use an existing version.
<a href="#">UpdateRelations</a>	Update in accordance with the current filter settings .
<a href="#">ResetItem</a>	Reset the script item to its initial state.
<a href="#">DeleteVersionForced</a>	Force deletion for non-leaf versions.

**Note**

*The versioning concept will probably be subjected to a major change in a later release. You must take into account to have to (ex-)change scripts using this interface.*

## 5.10.1 List of XScriptItemVersion Methods

### 5.10.1.1 Create

**SYNTAX****Create(bstrOldBDOObjectId)**

Parameter	Description
bstrOldBDOObjectId	The object of which a new version should be created.

**Script Return Value**

The object ID of the newly created version.

**Example****Example**

```
REM This script creates and uses a new version of the object
REM from which the script is executed
sub main(old_sci_id)
    old_base_id = Data.GetAttributeById(old_sci_id, "ergocomp-
base")
    old_base_name = Data.GetAttributeById(old_sci_id, "name")
    new_version_id = Version.Create(old_base_id)
    if new_version_id <> "" then
        MsgBox("New version of <" & old_base_name & "> created.")
        parent_id = Data.GetAttributeById(old_sci_id,
"ergocompbase_parent")
        if parent_id <> "" then
            call Version.Use(new_version_id, parent_id, old_sci_id)
            MsgBox("New version of <" & old_base_name & "> used.")
        end if
    else
        MsgBox("Checkout failed.")
    end if
end sub
```

### 5.10.1.2 CreateDeep

**SYNTAX****Create(bstrOldBDOObjectId)**

Parameter	Description
bstrOldBDOObjectId	The object of which a new version should be created.

**Script Return Value**

The object ID of the newly created version.

**Example****Example**

```

REM This script creates and uses a new version of the object
REM from which the script is executed
sub main(old_sci_id)
  old_base_id = Data.GetAttributebyId(old_sci_id, "ergocomp-
base")
  old_base_name = Data.GetAttributebyId(old_sci_id, "name")
  new_version_id = Version.CreateDeep(old_base_id)
  if new_version_id <> "" then
    MsgBox("New version of <" & old_base_name & "> created.")
    parent_id = Data.GetAttributebyId(old_sci_id,
"ergocompbase_parent")
    if parent_id <> "" then
      call Version.Use(new_version_id, parent_id, old_sci_id)
      MsgBox("New version of <" & old_base_name & "> used.")
    end if
  else
    MsgBox("Checkout failed.")
  end if
end sub

```

Works similar to [Create](#), however, this method guarantees that children components (recursion) are automatically versioned, too.

**5.10.1.3 Use****SYNTAX**

**Use(bstrNewBDOObjectId, bstrParentObjectId, bstrOldUsageId)**

Parameters	Description
bstrNewBDOObjectId	The new version of the object to use instead of the old version.
bstrParentObjectId	The object id of the parent in whose childlist the old version should be replaced by the new one.
bstrOldUsageId	The object id of the usage object (subcompitem, relationship object) of the old version.

**Script Return Value**

None

**Example****Example**

```

REM This skript creates and uses a new version of the object
REM from which the skript is executed

sub main(old_sci_id)
  old_base_id = Data.GetAttributebyId(old_sci_id, "ergocompbase")
  old_base_name = Data.GetAttributebyId(old_sci_id, "name")
  new_version_id =Version.Create(old_base_id)

  if new_version_id <> "" then
    MsgBox("New version of <" & old_base_name & "> created.")
    parent_id = Data.GetAttributebyId(old_sci_id, "ergocomp-
base_parent")
    if parent_id <> "" then
      call Version.Use(new_version_id, parent_id, old_sci_id)
      MsgBox("New version of <" & old_base_name & "> used.")
    end if
  else
    MsgBox("Checkout failed.")
  end if
end sub

```

**5.10.1.4 UpdateRelations**

**SYNTAX****UpdateRelations(BSTR bstrObjectId)**

Parameters	Description
bstrObjectId	The object whose structure should be updated according to the filter setting.
bstrExtendedEffectivity	The given extended effectivity.

**Script Return Value**

None

**Example****Example**

REM Get the first version of an object  
 REM This script must be called on a subcompitem or a relationship object.  
 REM When the method is called, components and relations visible  
 REM in accordance with the current filter settings will be updated.

**sub main(old\_sci\_id)**

```

old_base_id = Data.GetAttributeById(old_sci_id, "ergocompbase")
if old_base_id <> "" then
  old_parent_id = Data.GetAttributeById(old_sci_id, "relationobject1")
  if old_parent_id <> "" then
    first_version_id = Version.GetFirstVersion(old_base_id)
    if first_version_id <> "" then
      call Version.Use(first_version_id, old_parent_id, old_sci_id)
      MsgBox("First Version used.")
      call Version.UpdateRelations(first_version_id, "#DROP(R(2))")
    end if
  end if
end if
end sub

```

**5.10.1.5 CheckOut****SYNTAX****CheckOut(bstrOldBDOObjectId, bstrParentObjectId, bstrOldUsageObjectId)**

Parameters	Description
bstrOldBDOObjectId	The object of which a new version should be created.
bstrParentObjectId	The object id of the parent in whose childlist the old version should be replaced by the new one.
bstrOldUsageObjectId	The object id of the usage object (subcompitem, relationship object) of the old version.

**Script Return Value**

The object ID of the newly created version.

**Example****Example**

REM This method checks out a new version of the object  
 REM from which the skript is executed.  
 REM Check out means creating a new version and use this version.  
 REM Note:: the in-parameters of the Version.CheckOut method have  
 REM to follow the rules:  
 REM old\_base\_id : ergocompbase of the object from which the  
 REM skript is executed  
 REM old\_sci\_id : the object (subcompitem, relationship object)  
 REM from which the skript is executed  
 REM parent\_id : the parent of the object from which the skript  
 REM is executed  
 REM The same holds for relationship objects

```

sub main(old_sci_id)
  parent_id = Data.GetAttributeById(old_sci_id, "ergocomp-
base_parent")

```

```

    old_base_id = Data.GetAttributebyId(old_sci_id, "ergocomp-
base")
    old_base_name = Data.GetAttributebyId(old_base_id, "name")
    new_version_id=Version.CheckOut(old_base_id,parent_id,old_sci
_id)>
    if new_version_id <> "" then
        MsgBox("Checked out a new version of <" & old_base_name &
">.")
    else
        MsgBox("Checkout failed.")
    end if
end sub

```

**Note**

Checkout means [Create](#) and a subsequent [Use](#).

**5.10.1.6 CheckOutDeep****SYNTAX**

**CheckOut(bstrOldBDOObjectId, bstrParentObjectId,  
bstrOldUsageId)**

Parameters	Description
bstrOldBDOObjectId	The object of which a new version should be created.
bstrParentObjectId	The object id of the parent in whose childlist the old version should be replaced by the new one.
bstrOldUsageId	The object id of the usage object (subcompitem, relationship object) of the old version.

**Script Return Value**

The object ID of the newly created version.

**Example****Example**

```

REM This method checks out a new version of the object
REM from which the skript is executed.
REM Check out means creating a new version and use this version.
REM Note:: the in-parameters of the Version.CheckOut method have
REM to follow the rules:
REM old_base_id : ergocompbase of the object from which the
REM skript is executed
REM old_sci_id : the object (subcompitem, relationship object)
REM from which the skript is executed
REM parent_id : the parent of the object from which the skript
REM is executed
REM The same holds for relationship objects
sub main(old_sci_id)
    parent_id = Data.GetAttributebyId(old_sci_id, "ergocomp-
base_parent")
    old_base_id = Data.GetAttributebyId(old_sci_id, "ergocomp-
base")
    old_base_name = Data.GetAttributebyId(old_base_id, "name")
    new_version_id=Version.CheckOut(old_base_id,parent_id,old_sci
_id)>
    if new_version_id <> "" then
        MsgBox("Checked out a new version of <" & old_base_name &
">.")
    else
        MsgBox("Checkout failed.")
    end if
end sub

```

**Note**

Works similar to [CheckOut](#), however, this method guarantees that children components (recursion) are automatically versioned, too. Checkout means [CreateDeep](#) and a subsequent [Use](#).

**5.10.1.7 SetPlanningState****SYNTAX**

**SetPlanningState(bstrObjectId, bstrPlanningStateObjectId)**

Parameters	Description
bstrObjectId	The version whose planning state should be (re-)set.
bstrPlanningStateObjectId	The object ID of planning state object to which the version should be set.

**Script Return Value**

none

**Example****Example**

```
REM Setting the planning state of a component.
REM Call this skript on the component for which you want to (re-)
REM set the planning state.
REM Make sure to use the "right" planning state by using a man-
REM datory/unique attribute in the query.
sub main (id)
    object_id=Data.GetAttributeById(id, "ergocompbase")
    ps_name="Int1"
    project_id=Data.GetAttributeById(object_id, "ergoproject")
    if project_id <> "" then
        REM Find the id of the planning State named ps_name
        call Query.ResetSearch
        call Query.SetQuery("planningstate", "m_pDODefaultImpl",
"=", project_id)
        call Query.SetConcatenator("AND")
        call Query.SetQuery("planningstate", "name", "=", ps_name)
        REM Expecting a single result if name of ps is mandatory
        result_id=Query.GetOnlyFirstResult
        if result_id <> "" then
            call Version.SetPlanningState(object_id, result_id)
        else
            MsgBox("No planning state found.")
        end if
    else
        MsgBox("Could not retrieve project.")
    end if
end sub
```

**5.10.1.8 GetPlanningState****SYNTAX**

**GetPlanningState(bstrObjectId)**

Parameter	Description
bstrObjectId	The version whose planning state should be obtained.

**Script Return Value**

The object ID of the planning state object that is obtained.

**Example****Example**

```
sub main(comp_id)
    comp_name = Data.GetAttributeById(comp_id, "name")
    comp_base_id=Data.GetAttributeById(comp_id, "relationobject2")
    ps_id = Version.GetPlanningState(comp_base_id)
```

```

If ps_id <> "" then
    ps_name = Data.GetAttributebyId(ps_id, "name")
    call Dialog.MessageBox("Info", "Planning state of " + _
        comp_name + " is: " + ps_name)
End if
End sub

```

### 5.10.1.9 GetPlanningStates

#### SYNTAX

**GetPlanningStates(bstrObjectId, bstrMode, bstrScope)**

Parameters	Description
bstrObjectId	Any object ID within the project.
bstrMode	Restricts the returned planning states to an area or a higher level.
bstrScope	Retrieve the planning states global or local.

Possible values for **bstrMode**

Value	Description
"ALL"	Obtain all planning states.
"WORKING"	Obtain planning states of state "working" only.
"INTEGRATE"	Obtain planning states of state "integrate" only.
"RELEASED"	Obtain planning states of state "released" only.
"HIGHER_LEVEL"	Obtain planning states of a state "higher" than the current only.

Possible values for **bstrScope**

Value	Description
"LOCAL"	Obtain planning states from the <b>project</b> library.
"GLOBAL"	Obtain planning states from the library.

#### Script Return Value

An array containing the ID's of the obtained planning state objects.

#### Example

#### Example

```

sub main(sci_id)
    base_id = Data.GetAttributebyId(sci_id, "relationobject2")
    states=Version.GetPlanningStates(base_id, "ALL", "LOCAL")
    numberofstates = ubound(states,1)
    if numberofstates >= 0 then
        for i = 0 to numberofstates
            state_id = states( i )
            if state_id <> "" then
                ps_name = Data.GetAttributebyId(state_id, "name")
                MsgBox("PlanningState " & cstr(i) & "--> " & ps_name)
            end if
        next
    else
        MsgBox("No planning states found.")
    end if
end sub

```

### 5.10.1.10 GetFirstVersion

#### SYNTAX

**GetFirstVersion (bstrObjectId)**

Parameter	Description
bstrObjectId	The object of which the first version should be retrieved.

#### Script Return Value



The object ID of the first version.

### Example

#### Example

```
REM Get the first version of an object
REM This skript must be called on a subcompitem or a relation-
ship object.
sub main(old_sci_id)
old_base_id = Data.GetAttributebyId(old_sci_id, "ergocompbase")
if old_base_id <> "" then
  old_parent_id = Data.GetAttributebyId(old_sci_id, "ergocomp-
base_parent")
  if old_parent_id <> "" then
    first_version_id=Version.GetFirstVersion(old_base_id)
    if first_version_id <> "" then
      call Version.Use(first_version_id, old_parent_id,
old_sci_id)
      MsgBox("First Version used.")
    end if
  end if
end if
end sub
```

### 5.10.1.11 CreateReleaseTable

#### SYNTAX

**CreateReleaseTable (bstrOldBDOObjectId);**

Parameter	Description
bstrOldBDOObjectId	The object of which a new table should be created.

#### Script Return Value

The object ID of the newly created table.

#### Example

### Example

```
REM This script creates and uses a new release table of the ob-
ject
REM from which the script is executed
sub main(old_sci_id)

old_base_id = Data.GetAttributebyId(old_sci_id, "relationob-
ject2")
old_base_name = Data.GetAttributebyId(old_sci_id, "name")
new_releasetable_id = Version.CreateReleaseTable(old_base_id)

if new_releasetable_id <> "" then
MsgBox("New release table of <" & old_base_name & "> created.")
else
MsgBox("Creation failed.")
end if
end sub
```



#### Note

*Creation of a release table. If there is already a release table for the object version, the existing table is returned instead and a note is written into the log file.*

### 5.10.1.12 GetReleaseTable

#### SYNTAX

**GetReleaseTable (bstrOldBDOObjectId)**

Parameter	Description
bstrOldBDOObjectId	The object of which to get a release table.

**Script Return Value**

The object ID of the release table.

**Example****Example**

```
REM This script gets the existing release table of the object
REM from which the script is executed
sub main(old_sci_id)

old_base_id = Data.GetAttributebyId(old_sci_id, "relationob-
ject2")
old_base_name = Data.GetAttributebyId(old_sci_id, "name")
existing_releasetable_id = Version.GetReleaseTable(old_base_id)

if existing_releasetable_id <> "" then
MsgBox("Existing Release Table is " & existing_releasetable_id &
" .")
else
MsgBox("GetReleaseTable failed.")
end if
end sub
```

**Note**

*Allocating an existing release table.*

**5.10.1.13 CreateSFI****SYNTAX****CreateSFI (bstrOldBDOObjectld)**

Parameter	Description
bstrOldBDOObjectld	The object of which a new version should be created.

**Script Return Value**

The object ID of the newly created version.

**Example****Example**

```
REM This script creates and uses a new version of the object
REM from which the script is executed
sub main(old_sci_id)

old_base_id = Data.GetAttributebyId(old_sci_id, "relationob-
ject2")
old_base_name = Data.GetAttributebyId(old_sci_id, "name")
new_version_id = Version.CreateSFI(old_base_id)

if new_version_id <> "" then
MsgBox("New version of <" & old_base_name & "> created.")
parent_id = Data.GetAttributebyId(old_sci_id, "relationobject1")
if parent_id <> "" then
call Version.Use(new_version_id, parent_id, old_sci_id)
MsgBox("New version of <" & old_base_name & "> used.")
end if
else
MsgBox("Checkout failed.")
end if
end sub
```

**5.10.1.14 CreateDeepSFI****SYNTAX****CreateSFI (bstrOldBDOObjectld)**

Parameter	Description
bstrOldBDOObjectId	The object of which a new version should be created.

**Script Return Value**

The object ID of the newly created version.

**Example****Example**

```
REM This script creates and uses a new version of the object
REM from which the script is executed
sub main(old_sci_id)

old_base_id = Data.GetAttributebyId(old_sci_id, "relationob-
ject2")
old_base_name = Data.GetAttributebyId(old_sci_id, "name")
new_version_id = Version.CreateDeepSFI(old_base_id)

if new_version_id <> "" then
MsgBox("New version of <" & old_base_name & "> created.")
parent_id = Data.GetAttributebyId(old_sci_id, "relationobject1")
if parent_id <> "" then
call Version.Use(new_version_id, parent_id, old_sci_id)
MsgBox("New version of <" & old_base_name & "> used.")
end if
else
MsgBox("Checkout failed.")
end if
end sub
```

**5.10.1.15 CheckOutSFI****SYNTAX**

**CheckOutSFI (bstrOldOObjectId, bstrParentObjectId, bstrOldUsageObjectl)**

Parameters	Description
bstrOldBDOObjectId	The object of which a new version should be created.
bstrParentObjectId	The object id of the parent in whose childlist the old version should be replaced by the new one.
bstrOldUsageObjectId	The object id of the usage object (subcompitem, relationship object) of the old version.

**Script Return Value**

The object ID of the newly created version.

**Example****Example**

```
REM This method checks out a new version of the object
REM from which the script is executed
REM Check out means creating a new version and use this version.
REM Note: the in-parameters of the Version.CheckOut method have
to follow the rules:
REM old_base_id : ergocompbase of the object from which the
script is executed
REM old_sci_id : the object (subcompitem, relationship object)
from which the script is executed
REM parent_id : the parent of the object from which the script
is executed
REM The same holds for relationship objects
sub main(old_sci_id)

parent_id = Data.GetAttributebyId(old_sci_id, "relationobject1")
old_base_id = Data.GetAttributebyId(old_sci_id, "relationob-
ject2")
```

```

old_base_name = Data.GetAttributebyId(old_base_id, "name")

new_version_id=Version.CheckOutSFI(old_base_id, parent_id,
old_sci_id)
if new_version_id <> "" then
MsgBox("Checked out a new version of <" & old_base_name & ">.")
else
MsgBox("Checkout failed.")
end if
end sub

```

### 5.10.1.16 CheckOutDeepSFI

#### SYNTAX

**CheckOutDeepSFI (bstrOldOObjectId, bstrParentObjectId, bstrOldUsageObjectl)**

Parameters	Description
bstrOldBDOObjectId	The object of which a new version should be created.
bstrParentObjectId	The object id of the parent in whose childlist the old version should be replaced by the new one.
bstrOldUsageObjectId	The object id of the usage object (subcompitem, relationship object) of the old version.

#### Script Return Value

The object ID of the newly created version.

#### Example

#### Example

```

REM This method checks out a new version of the object
REM from which the script is executed
REM Check out means creating a new version and use this version.
REM Note: the in-parameters of the Version.CheckOut method have
REM to follow the rules:
REM old_base_id : ergocompbase of the object from which the
REM script is executed
REM old_sci_id : the object (subcompitem, relationship object)
REM from which the script is executed
REM parent_id : the parent of the object from which the script
REM is executed
REM The same holds for relationship objects
sub main(old_sci_id)

parent_id = Data.GetAttributebyId(old_sci_id, "relationobject1")
old_base_id = Data.GetAttributebyId(old_sci_id, "relationob-
ject2")
old_base_name = Data.GetAttributebyId(old_base_id, "name")

new_version_id=Version.CheckOutDeepSFI(old_base_id, parent_id,
old_sci_id)
if new_version_id <> "" then
MsgBox("Checked out a new version of <" & old_base_name & ">.")
else
MsgBox("Checkout failed.")
end if
end sub

```

### 5.10.1.17 PDXMLFileCreated

#### SYNTAX

**PDXMLFileCreated (bstrOldOObjectId, bstrExtendedEffectivity, bWasCreated)**

Parameters	Description
bstrOldBDOObjectId	The object of which a new version should be created.
bstrExtendedEffectivity	Extended Effectivity
bWasCreated	"True"

**Script Return Value**

None

**Example****Example**

```

option explicit
public sub main(old_sci_id)

    dim WshShell
    Set WshShell = CreateObject("WScript.Shell")
    dim old_base_id
    dim old_parent_id
    dim release_table_id

    old_base_id = Data.GetAttributebyId(old_sci_id, "ergocompbase")
    if old_base_id <> "" then
        old_parent_id = Data.GetAttributebyId(old_sci_id, "ergocomp-
base_parent")
        if old_parent_id <> "" then
            call Version.PDXMLFileCreated(old_base_id , "{2-10}", true)
        end if
    end if
end sub

```

**5.10.1.18 Promote**

This method promotes a component to a higher planning state.

**SYNTAX****Promote()**

Parameter	Description
-	-

**Script Return Value**

none

**Example****Example**

```

Sub main(id)
    ...
    Version.Promote(id)
    ...
End sub

```

**5.10.1.19 CanPlanningStateBeModified****SYNTAX****CheckOut (bstrOldBDOObjectId)**

Parameters	Description
bstrOldBDOObjectId	The component whose modifiability is to check.

**Script Return Value**

Modifiability Value

**Example****Example**

```

REM This script checks a components' modifiability
sub main(old_sci_id)
    old_base_id = Data.GetAttributebyId(old_sci_id, "ergocompbase")
    old_base_name = Data.GetAttributebyId(old_sci_id, "name")
    modifiable = Version.CanPlanningStateBeModified( old_base_id)

```

```
MsgBox( old_base_name & ": Modifiability is " & modifiable)
End sub
```

### 5.10.1.20 DeleteVersionForced

#### SYNTAX

**DeleteVersionForced (bstrOldBDOObjectId)**

Parameters	Description
bstrOldBDOObjectId	The version to delete.

```
HRESULT DeleteVersionForced( [in] BSTR
bstrOldBDOObjectId);
```

#### VTABLE Return Value

##### HRESULT

Parameters	Description
S_OK	If successful.
Other Error Codes	If deletion fails.

#### Script Return Value

HRESULT value

\* S\_OK on success

\* E\_VERSION\_DELETE\_IN\_HISTORY\_NEEDS\_FUNCTIONRIGHT if current user has not function right "versioninhistory\_allow\_delete"

\* E\_VERSION\_VERSIONING\_NOT\_ALLOWED\_FOR\_THIS\_TYPE if associated planning state prevents deletion and user has not function right "version\_allow\_delete"

\* E\_VERSION\_DELETE\_FIRSTVERSION\_NOT\_LEAF if trying to delete first version that has successor versions.

\* E\_ on failure for other failures

#### Example

##### Example

```
Sub main_id
  call version.DeleteVersionForced(id)
End sub
```

### 5.10.1.21 DeleteVersionForcedEx

#### SYNTAX

**DeleteVersionForcedEx(objectID,bool bCCZCheck)**

Parameters	Description
objectID	Object ID of the component to be deleted.
bCCZCheck	<p>"bCCZCheck" is the Boolean variable to decide if API should respect the CCZ.</p> <p>If bCCZCheck = 0 the API will respect the CCZ and cannot be deleted.</p> <p>If bCCZCheck = 1 then the component irrespective of the CCZ membership option can be deleted.</p>

#### Example

##### Example

```
function sa_delete(object_id)
  name = Data.GetAttributebyId(object_id, "name")
  call version.DeleteVersionForcedEx(object_id,true)
  sa_delete=false
end function
```

The API method *DeleteVersionForcedEx()* for the interface *IEPVersion* allows users with a particular function right to delete CCZ Members and its CCZ children versions. *DeleteVersionForcedEx()* is monitored by a logging functionality.

*DeleteVersionForcedEx* allows to get rid of not needed CCZ member, previously versioned objects either manually through the automatically using scripting or using a C++ API. You can execute this API method either by creating Script Actions or by directly executing the script on the CCZ component.

### Function Rights

To use the new API method `IEPVersion::DeleteVersionForcedEx`, you must either have the function right “`versioninghistory_allow_delete`” or be a super user. (i.e. all the rights for API `DeleteVersionForced` will be valid for `DeleteVersionForcedEx`).

If the corresponding function right is not granted, the attempt to delete will yield error code to the

“`E_VERSION_DELETE_IN_HISTORY_NEEDS_FUNCTIONRIGHT`”.

Since the “**versions\_allow\_delete**” function right allows deletion of versions regardless of its planning state, the calling of `IEPVersion::DeleteVersionForcedEx` might additionally need this function right, as well.

The first version can only be deleted, when it is the only version in its version history or if it has only one successor version. An attempt to delete the first version having multiple successor versions will yield to the `E_VERSION_DELETE_FIRSTVERSION_HAS_MULTIPLE_SUCCESSORS` error code.

### IPDserver\_DeleteVersionForced log file

The versions will be permanently deleted from the database, for information tracking purposes a log file will be written, while versions are deleted.

The name of the log file is *IPDServer\_DeleteVersionForced.log* (i.e. the log file for `DeleteVersionForced` and `DeleteVersionForcedEx` will be logged in same log file). The log file is located in the PPR Server log directory. For each successful deletion, a log entry containing name and version number of the deleted version as well as name of the logged-in user executing the deletion is written to that log file. The log file logged additionally the name and the version number of the first, predecessor, and redirected successor versions.

The deletion of leaf versions through `DeleteVersionForcedEx` is also written in the log file.

Deletions of these versions are irrespective of the selected Action, Mod statement, and the Planning states.



### Note

*The first version can only be deleted if it has not more than one successor version, i.e. The version cannot be deleted if there exists parallel versions.*

*Be careful while linking the previous versions of deleted CCZ Member in order not to share different CCZ Owners.*

## 5.11 Class ScriptItemUnit

The [Table 21](#) describes class scriptitem unit methods.

**Table 21: Class ScriptItem Unit Methods**

Method	Description
<a href="#">GetDefaultUnitByAttr</a>	Retrieve default unit from type name and attribute name (key).
<a href="#">GetDefaultUnitByCat</a>	Retrieve default unit by unit category.
<a href="#">GetDefaultUnitByUnit</a>	Retrieve default unit by other unit of this category.
<a href="#">GetUIDefaultUnitByAttr</a>	Retrieve user interface (UI) default unit from type name and attribute name (key).
<a href="#">GetUIDefaultUnitByCat</a>	Retrieve user interface (UI) default unit by unit category.
<a href="#">GetUIDefaultUnitByUnit</a>	Retrieve user interface (UI) default unit by other unit of this category.
<a href="#">GetAllUnitsByAttr</a>	Get an array of all units of a category from type name and attribute name (key).
<a href="#">GetAllUnitsByCat</a>	Get an array of all units of a category.
<a href="#">Convert</a>	Convert a value from source to target unit.
<a href="#">ConvertToDefaultUnit</a>	Convert a value from source to default unit.
<a href="#">ConvertToUIDefaultUnit</a>	Convert a value from source to user interface (UI) default unit.
<a href="#">ResetItem</a>	Reset the script item to its initial state.

### 5.11.1 List of XScriptItemUnit Methods

#### 5.11.1.1 GetDefaultUnitByAttr

##### SYNTAX

**GetDefaultUnitByAttr(bstrTypeName, bstrAttributeName)**

Parameters	Description
bstrTypeName	A unique typename.
bstrAttributeName	The name of an attribute of this type (typename AND attributename define a unique key).

##### Script Return Value

Parameter	Description
bstrDefUnit	The default unit (e.g. "UA_TIME_SEC").

#### Example

```
sub main(id)
...
type_name = "ergocompprocessdefault"
attribute_name = "time"
def_unit = Unit.GetDefaultUnitbyAttr(type_name, attribute_name)
...
end sub
```



#### Note

Each attribute belongs to a category (For example, "UA\_TIME"). The default unit (i.e. "UA\_TIME\_SEC") states, in which unit the attribute values of an attribute of a type are really stored. The user interface (UI) default unit of this attribute can be different from this database default unit and moreover one can configure the current unit (For example, "UA\_TIME\_MIN"), in which the attribute is displayed in properties dialog, browser etc.

#### 5.11.1.2 GetDefaultUnitByCat

##### SYNTAX

**GetDefaultUnitByCat(bstrCategory)**



Parameter	Description
bstrCategory	The unit category (For example, "UA_TIME").

**Script Return Value**

Parameter	Description
bstrDefUnit	The default unit (For example, "UA_TIME_SEC").

**Example****Example**

```
sub main(id)
...
unitcategory = "UA_TIME"
def_unit = Unit.GetDefaultUnitbyCat(unitcategory)
...
end sub
```

**Note**

Each attribute belongs to a category (For example, "UA\_TIME"). The default unit (i.e. "UA\_TIME\_SEC") states, in which unit the attribute values of an attribute of a type are really stored. The user interface (UI) default unit of this attribute can be different from this database default unit and moreover one can configure the current unit (For example, "UA\_TIME\_MIN"), in which the attribute is displayed in properties dialog, browser etc.

**5.11.1.3 GetDefaultUnitByUnit****SYNTAX****GetDefaultUnitByUnit(bstrUnit)**

Parameter	Description
bstrUnit	Any other unit (e.g. "UA_TIME_MIN") belonging to the same unit category (For example, "UA_TIME").

**Script Return Value**

Parameter	Description
bstrDefUnit	The default unit (For example, "UA_TIME_SEC").

**Example****Example**

```
sub main(id)
...
currentunit = "UA_TIME_DAY"
def_unit = Unit.GetDefaultUnitbyUnit(currentunit)
...
end sub
```

**Note**

Each attribute belongs to a category (For example, "UA\_TIME"). The default unit (i.e. "UA\_TIME\_SEC") states, in which unit the attribute values of an attribute of a type are really stored. The user interface (UI) default unit of this attribute can be different from this database default unit and moreover one can configure the current unit (For example, "UA\_TIME\_MIN"), in which the attribute is displayed in properties dialog, browser etc.

**5.11.1.4 GetUIDefaultUnitByAttr****SYNTAX****GetUIDefaultUnitByAttr(bstrTypeName, bstrAttributeName)**

Parameters	Description
bstrTypeName	A unique typename.
bstrAttributeName	The name of an attribute of this type (typename AND attributename)

Parameters	Description
	define a unique key).

**Script Return Value**

Parameters	Description
bstrDefUnit	The UI default unit (i.e. "UA_TIME_SEC").

**Example****Example**

```
sub main(id)
...
type_name = "ergocompprocessdefault"
attribute_name = "time"
def_unit = Unit.GetUIDefaultUnitbyAttr(type_name, attribute_name)
...
end sub
```

**Note**

Each attribute belongs to a category (For example, "UA\_TIME"). The default unit (i.e. "UA\_TIME\_SEC") states, in which unit the attribute values of an attribute of a type are really stored. The user interface (UI) default unit of this attribute can be different from this database default unit and moreover one can configure the current unit (For example, "UA\_TIME\_MIN"), in which the attribute is displayed in properties dialog, browser etc.

**5.11.1.5 GetUIDefaultUnitByCat****SYNTAX****GetUIDefaultUnitByCat(bstrCategory)**

Parameter	Description
bstrCategory	The unit category (i.e. "UA_TIME").

**Script Return Value**

Parameter	Description
bstrDefUnit	The UI default unit (i.e. "UA_TIME_SEC").

**Example****Example**

```
sub main(id)
...
unitcategory = "UA_TIME"
def_unit = Unit.GetUIDefaultUnitbyCat(unitcategory)
...
end sub
```

**Note**

Each attribute belongs to a category (For example, "UA\_TIME"). The default unit (i.e. "UA\_TIME\_SEC") states, in which unit the attribute values of an attribute of a type are really stored. The user interface (UI) default unit of this attribute can be different from this database default unit and moreover one can configure the current unit (For example, "UA\_TIME\_MIN"), in which the attribute is displayed in properties dialog, browser etc.

**5.11.1.6 GetUIDefaultUnitByUnit****SYNTAX****GetUIDefaultUnitByUnit(bstrUnit)**

Parameter	Description
bstrUnit	Any other unit (For example, "UA_TIME_MIN") belonging to the same unit category (i.e. "UA_TIME").

**Script Return Value**

Parameter	Description
bstrDefUnit	The UI default unit (i.e. "UA_TIME_SEC").

**Example****Example**

```
sub main(id)
...
currentunit = "UA_TIME_DAY"
def_unit = Unit.GetUIDefaultUnitbyUnit(currentunit)
...
end sub
```

**Note**

Each attribute belongs to a category (For example, "UA\_TIME"). The default unit (i.e. "UA\_TIME\_SEC") states, in which unit the attribute values of an attribute of a type are really stored. The user interface (UI) default unit of this attribute can be different from this database default unit and moreover one can configure the current unit (For example, "UA\_TIME\_MIN"), in which the attribute is displayed in properties dialog, browser etc.

**5.11.1.7 GetAllUnitsByAttr****SYNTAX**

**GetAllUnitsByAttr(bstrTypeName, bstrAttributeName)**

Parameters	Description
bstrTypeName	A unique typename.
bstrAttributeName	The name of an attribute of this type (typename AND attributename define a unique key).

**Script Return Value**

Parameters	Description
pUnits	An array of pairs of visible unit name (For example, "seconds") and internal unit name (i.e. "UA_TIME_SEC").

**Example****Example**

```
sub main(id)
...
type_name = "ergocompprocessdefault"
attribute_name = "time"
units = Unit.GetAllUnitsByAttr(type_name, attribute_name)
Set d = CreateObject("Scripting.Dictionary")
dimunits = ubound(units, 1)
REM write the visible/internal values to a VB dictionary
if dimunits >= 0 then
for m = 0 to dimunits
d.Add units(m, 0), units(m, 1)
next
end if
...
end sub
```

**Note**

The internal unit name (For example, "UA\_TIME\_SEC") is used in other function calls - not the visible unit name (i.e. "seconds").

**5.11.1.8 GetAllUnitsByCat****SYNTAX**

**GetAllUnitsByCat(bstrCategory)**

Parameter	Description
bstrCategory	The unit category (For example, UA_TIME").

**Script Return Value**

Parameter	Description
pUnits	An array of pairs of visible unit name (For example, "seconds") and internal unit name (i.e. "UA_TIME_SEC").

**Example****Example**

```

sub main(id)
...

unitcategory = "UA_TIME"

units = Unit.GetAllUnitsByCat(unitcategory)

Set d = CreateObject("Scripting.Dictionary")
dimunits = ubound(units, 1)
REM write the visible/internal values to a VB dictionary
if dimunits >= 0 then
for m = 0 to dimunits
d.Add units(m, 0), units(m, 1)
next
end if
...
end sub

```

**Note**

*The internal unit name (For example, "UA\_TIME\_SEC") is used in other function calls - not the visible unit name (i.e. "seconds").*

**5.11.1.9 Convert****SYNTAX**

**Convert(bstrSourceUnit, bstrTargetUnit, vSourceValue)**

Parameters	Description
bstrSourceUnit	The source unit.
bstrTargetUnit	The target unit.
vSourceValue	The value before conversion to the target unit.

**Script Return Value**

Parameter	Description
vTargetValue	The value after conversion to the target unit.

**Example****Example**

```

sub main(id)
...
currentunit = "UA_TIME_DAY"
selectedunit = "UA_TIME_SEC"
inputvalue = 1
outputvalue = Unit.Convert (currentunit, selectedunit, inputvalue)
...
end sub

```

**5.11.1.10 ConvertToDefaultUnit****SYNTAX**

**ConvertToDefaultUnit(bstrSourceUnit, vSourceValue)**

Parameters	Description
bstrSourceUnit	The source unit.

Parameters	Description
vSourceValue	The value before conversion to the default unit.

**Script Return Value**

Parameter	Description
vTargetValue	The value after conversion to the default unit.

**Example****Example**

```
sub main(id)
...
currentunit = "UA_TIME_DAY"
inputvalue = 1
outputvalue = Unit.ConvertToDefaultUnit (currentunit, inputvalue)
...
end sub
```

**5.11.1.11 ConvertToUIDefaultUnit****SYNTAX****ConvertToUIDefaultUnit(bstrSourceUnit, vSourceValue)**

Parameters	Description
bstrSourceUnit	The source unit.
vSourceValue	The value before conversion to the UI default unit.

**Script Return Value**

Parameter	Description
vTargetValue	The value after conversion to the UI default unit.

**Example****Example**

```
sub main(id)
...
currentunit = "UA_TIME_DAY"
inputvalue = 1
outputvalue = Unit.ConvertToUIDefaultUnit (currentunit, inputvalue)
...
end sub
```

**5.12 Class ScriptItemLock**

The [Table 22](#) describes the class scriptitem lock methods.

**Table 22: Class ScriptItem Lock Methods**

Method	Description
<a href="#">LockObject</a>	Lock an object
<a href="#">UnlockObject</a>	Unlock an object previously locked using LockObject.
<a href="#">UnlockAllObject</a>	Unlock all objects previously locked using LockObject in a single shot. IDs of locked objects are temporarily stored in a list, function call clears that list, too.
<a href="#">IsObjectLocked</a>	Check whether an object is already locked by somebody else.
<a href="#">GetObjectLockInfo</a>	Get Lock State Information.
<a href="#">GetObjectLockInfoAll</a>	Get lock information of all lockers except from the own client ID concerning a given data type.
<a href="#">ResetItem</a>	Reset the script item to its initial state.



## Caution

*Some users erroneously believed that it would be possible to unlock objects which have been locked by other users. This is of course not possible. Moreover, locks are not persistent. You can only unlock objects that were locked within the same transaction (client process) in which the script itself is running.*

## 5.12.1 List of XScriptItemLock Methods

### 5.12.1.1 LockObject

#### Syntax

**LockObject(bstrObjectId, bstrLockMode)**

Parameters	Description
bstrObjectId	The object to be locked.
bstrLockMode	The mode in which you would like the component to be locked (Please refer to the <a href="#">Note</a> ).

Possible values for **bstrLockMode** are:

- NOT\_DELETE
- NOT\_LINK
- READ
- LINK
- WRITE
- DELETE

#### Script Return Value

Parameters	Description
bFlag	True(-1): In case the lock command has been successful. False(0): otherwise (e.g. the component is already locked by someone else).

### Example

#### Example

```
sub main(id)
base_id = Data.GetAttributebyId(id, "ergocompbase")
isLocked = CompLock.IsObjectLocked(base_id)
if isLocked = True then
    info = CompLock.GetObjectLockInfo(base_id)
    MsgBox(info)
else
    lockit = CompLock.LockObject(base_id, "WRITE")
    if lockit = True then
        MsgBox("Lock successful.")
    else
        MsgBox("Lock not successful.")
    end if
end if
end sub
```

### 5.12.1.2 UnlockObject

#### Syntax

**UnlockObject(bstrObjectId, bstrLockMode, blmImmediateCommit)**

Parameters	Description
bstrObjectId	The object to be locked.
bstrLockMode	The mode in which you would like the component to be unlocked (see Note).
blmmediateCommit	A flag indicating if you like to commit the unlock immediately. If you do so, other users can immediately edit the unlocked component, otherwise they have to wait for your next commit.

### Note



*It is strongly recommended NOT to use this option with flag '1'. It is hard to imagine a use case, which justifies using the option 'blmmediateCommit = 1'. Moreover, blmmediateCommit must not be set to '1' if an object has actually been changed. Misuse may cause database deadlocks.*

Script Return Valuee for **bstrLockMode** are:

- NOT\_DELETE
- NOT\_LINK
- READ
- LINK
- WRITE
- DELETE

### Script Return Value

Parameters	Description
bFlag	True(-1): In case the unlock command has been successful. False(0): otherwise (e.g. the component cannot be unlocked).

### Example

#### Example

```
sub main(id)
base_id = Data.GetAttributebyId(id, "ergocompbase")
'Immediate commit of unlock
iCommitUnLock = 0
unlockit = CompLock.UnlockObject(base_id, "WRITE", iCommitUn-
Lock)

if unlockit = True then
    MsgBox("Unlock successful")
else
    MsgBox("Unlock not successful. Object may not have been
locked at all.")
end if

end sub
```

### 5.12.1.3 UnlockAllObject

#### Syntax

**UnlockAllObject(blmediateComit)**

Parameter	Description
blmediateCommit	A flag indicating if you like to commit the unlock immediately. If you do so , other users can immediately edit the unlocked components, otherwise they have to wait for your next commit.

**Note**

*It is strongly recommended NOT to use this option with flag '1'. It is hard to imagine a use case, which justifies using the option 'blmmediateCommit = 1'. Moreover, blmmediateCommit must not be set to '1' if an object has actually been changed. Misuse may cause database deadlocks.*

**Script Return Value**

Parameter	Description
bFlag	True(-1): In case the unlock command has been successful. False(0): otherwise (e.g. the component cannot be unlocked).

**Example****Example**

```
sub main(notused)
...
unlockit = CompLock.UnlockAllObject(0)
...
end sub
```

**Note**

*The usage is similar to UnlockObject. It is assumed that all objects subjected to that function are locked using the same lock mode.*

**5.12.1.4 IsObjectLocked****SYNTAX****IsObjectLocked(bstrObjectId)**

Parameter	Description
bstrObjectId	The object ID.

**Script Return Value**

Parameter	Description
bFlag	True(-1): If the component is locked by someone else. False(0): Otherwise.

**Example****Example**

```
sub main(id)
base_id = Data.GetAttributebyId(id, "ergocombase")
isLocked = CompLock.IsObjectLocked(base_id)
if isLocked = True then
    info = CompLock.GetObjectLockInfo(base_id)
    MsgBox(info)
else
    lockit = CompLock.LockObject(base_id, "WRITE")
    if lockit = True then
        MsgBox("Lock successful.")
    else
        MsgBox("Lock not successful.")
    end if
end if
end sub
```

**Note**

*This methods tells you nothing about the lock state of the object. In case the object is locked you can get additional information using [GetObjectLockInfo](#).*

**5.12.1.6 GetObjectLockInfo**



**SYNTAX****GetObjectLockInfo(bstrObjectId)**

Parameter	Description
bstrObjectId	The object ID

The returned *plnfo* string contains the following information (separated by CRLF):

- Lock Mode (For example, "WRITE")
- User Name
- Computer Name
- Lock Date/Time

You can use this string and display it in a message box (like to be seen in the sample screenshot).

**Script Return Value**

Parameter	Description
plnfo	A string containing information about the lock state.

**Example****Example**

```

sub main(id)
base_id = Data.GetAttributebyId(id, "ergocompbase")
isLocked = CompLock.IsObjectLocked(base_id)
if isLocked = True then
    info = CompLock.GetObjectLockInfo(base_id)
    MsgBox(info)
else
    lockit = CompLock.LockObject(base_id, "WRITE")
    if lockit = True then
        MsgBox("Lock successful.")
    else
        MsgBox("Lock not successful.")
    end if
end if
end sub

```



Figure 40: VB Script

**5.12.1.7 GetObjectLockInfoAll****SYNTAX****GetObjectLockInfoAll (bstrObjectId, bstrDataType)**

Parameter	Description
bstrObjectId	The object ID
bstrDataType	Bstring containing data type

**Script Return Value**

Parameter	Description
bstrLockInfoAll	Pointer to BSTR containing lock information

**Example****Example** (script example to be called on an action)

```
sub main (id)
  info = CompLock.GetObjectLockInfoAll (id, "Action")
  if info = "" then
    MsgBox ("Object is not locked.")
  else
    MsgBox (info)
  end if
end sub
```

**Note**

*GetObjectLockInfoAll method is introduced from R18sp5 onwards.*

# List of Figures

Figure 1: Script Example.....	14
Figure 2: Interaction between the Script Engine and the DPE.....	16
Figure 3: Creating a New Script.....	18
Figure 4: Script Properties .....	19
Figure 5: User Management .....	20
Figure 6: Option for starting a Script .....	21
Figure 7: List of Available Scripts.....	22
Figure 8: New Script Action.....	23
Figure 9: Script Action Properties.....	23
Figure 10: New Script Action.....	26
Figure 11: Starting Script by Context Menu on Production Node .....	27
Figure 12: Creating Script Commands – Properties Dialog .....	27
Figure 13: Error Message for a Missing Valuation Method.....	30
Figure 14: Executing a Script or a VBA Macro in the Context Menu.....	30
Figure 15: Script Command Properties .....	31
Figure 16: Script Call-up in the Properties Dialog of an Object.....	32
Figure 17: Checking of Attribute.....	32
Figure 18: Script Assignment.....	33
Figure 19: Plantype Part Object .....	33
Figure 20: Display ID .....	35
Figure 21: Example of a Message with Usage of Scripts in a Script.....	41
Figure 22: Enabling Log Functions – Tools - Settings .....	42
Figure 23: Creating VBA Scripts – New Context Menu .....	43
Figure 24: VBA Project Properties Dialog .....	44
Figure 25: Opening VBA Environment – Context Menu .....	44
Figure 26: Example of a VBA Environment .....	45
Figure 27: Opening VBA Macros in the Context Menu .....	45
Figure 28: Macros Dialog – Project-Wide Selection .....	46
Figure 29: Macros Dialog – Project-Related Selection .....	46
Figure 30: Opening the Script Editor .....	47
Figure 31: VBA Application Property.....	48
Figure 32: VB Script.....	64
Figure 33: Get First Child.....	77
Figure 34: Abort/Retry/Cancel box.....	132
Figure 35: Example eines Dialog .....	134

Figure 36: FileSelector PE 5.7 .....	139
Figure 37: FileSelector PE 5.9 .....	140
Figure 38: RTF Formatted Text.....	171
Figure 39: VB Script.....	172
Figure 40: VB Script.....	195

## List of Tables

Table 1: Location of Scripts, VBA Macros, and Script Actions in PPR-Navigator .....	17
Table 2: Table with Entry Parameters with Script Actions .....	24
Table 3: Table for Actions .....	24
Table 4: Script Commands Entry Parameter .....	27
Table 5: Call Up Options .....	30
Table 6: Conversion of a Script into a Macro .....	47
Table 7: Description of the VBA Object Properties .....	48
Table 8: List of Script Actions Methods .....	50
Table 9: Class Documentation .....	69
Table 10: Class Script Item Data Methods .....	70
Table 11: Class SkriptItem Rights Methods .....	102
Table 12: Class Scriptitem Grid Methods .....	119
Table 13: Class Scriptitem Graphic Methods .....	120
Table 14: Identifier and File Names .....	123
Table 15: Class ScriptItem Dialog Methods .....	130
Table 16: Class ScriptItem List Methods .....	148
Table 17: Class ScriptItem Query Methods .....	150
Table 18: Class Scriptitem Config Methods .....	165
Table 19: Class ScriptItem Convert Methods .....	170
Table 20: Class ScriptItem Version Methods .....	172
Table 21: Class ScriptItem Unit Methods .....	185
Table 22: Class ScriptItem Lock Methods .....	191

# Index

## B

Batch Mode .....	32
Starting Scripts Automatically .....	32

## C

Class XscriptItemLock .....	
GetObjectLockInfo .....	190
IsObjectLocked .....	189
LockObject .....	187
UnlockObject .....	188, 189
Class XScriptItemLock .....	187
Concerned about Security when Using Scripts.....	14

## E

Error handling .....	
Syntax Errors.....	36
Error Handling .....	36
Run-time Errors .....	36

## F

Function .....	35
----------------	----

## I

Interactive Mode.....	21
-----------------------	----

## L

Log Functions.....	41
--------------------	----

## M

Main .....	35
------------	----

## N

New Functions .....	
PE 5.16SP4.....	3
PE 5.17SP2.....	2
Nonliability .....	ii

## O

object_id .....	35
-----------------	----

## P

Parameter .....	
Name.....	18
Number.....	18
Running as Owner.....	18
Running In Own Transaction.....	18
Script Language.....	18
Source Text .....	18
Placement of brackets.....	38
Procedure .....	35

## R

Real Script Actions.....	24
Recursions.....	37
Rights in Scripting.....	18
Run-Time Errors .....	36

## S

Script Actions .....	
in the Context Menu .....	30
Script Editor .....	17
ScriptItemRights.....	98
GetCurrentUser.....	100
GetUserFullName.....	100
Transfer .....	101
Transfer (PE 5.12).....	101
ScriptItemUnit .....	181
Scripts Created .....	17
Syntax Errors .....	36

## U

Unit .....	181
------------	-----

## V

VBA .....	41
VBA Environment.....	43