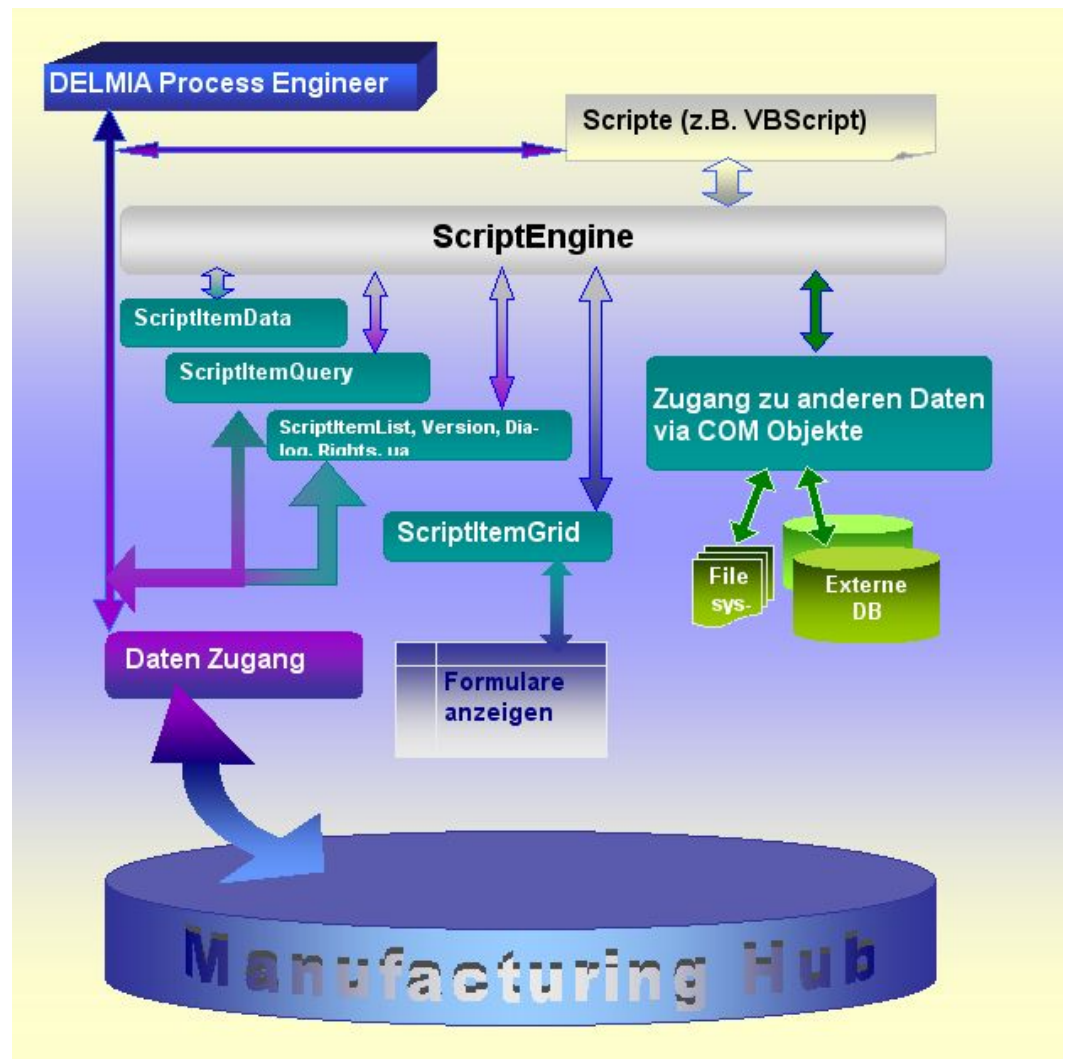




# Skripting



## Vorwort

Das vorliegende Handbuch führt Sie in die allgemeine Bedienung und die Erstellung von Skripten im DELMIA Process Engineer® ein.

Die Bedienung und Funktionsweise wird für Sie schnell und leicht erlernbar sein - eine benutzerfreundliche Bedienoberfläche und eine übersichtliche Menüführung erleichtert Planungsaufgaben schnell und sicher im Process Engineer durchzuführen.

Trotzdem wird es noch Sachverhalte geben, die wir noch verbessern können. Sollten Sie daher Vorschläge für Verbesserungen unserer Software haben, so lassen Sie uns dies bitte wissen.

Jede konstruktive Kritik ist uns willkommen. Denn sie hilft uns, die Arbeit mit dem Process Engineer für Sie noch einfacher und übersichtlicher zu gestalten.

Dasselbe gilt selbstverständlich auch für das vorliegende Handbuch. Wenn Sie an der einen oder anderen Stelle dieses Benutzerleitfadens das Gefühl haben, dass die Funktionen oder die Programmführung nicht ausreichend erklärt werden, wenden Sie sich bitte an Ihren direkten DELMIA-Ansprechpartner. Wir freuen uns auf Ihre Anmerkungen und Vorschläge.

### **Ausschluss jeder Haftung und Garantie**

Unsere Programme und Handbücher wurden mit großer Sorgfalt und nach bestem Wissen und Gewissen erstellt und entsprechend im Einsatz getestet. Jedoch wird keinerlei Haftung oder Gewähr dafür übernommen, dass die Software und die Beschreibungen fehlerfrei oder für spezielle Zwecke geeignet sind.

DELMIA übernimmt keine Haftung für sich aus der Verwendung dieser Software eventuell ergebende Schäden. Mit der Verwendung der Software erkennt der Benutzer diesen Haftungsausschluss an und stellt DELMIA von sämtlichen Ansprüchen frei.

### **Urheberrecht**

Alle in unseren Unterlagen enthaltenen Informationen dürfen für interne Zwecke gerne kopiert und weiter verwendet werden, solange dies kostenlos geschieht und die Inhalte nicht verändert oder verfälscht werden.

Jede andere Form der Nutzung, insbesondere der Vertrieb auf CD-ROM oder in anderen Publikationen, insgesamt oder in Teilen, ist nur nach vorheriger schriftlicher Zustimmung durch DELMIA zulässig.

Teile dieser Software sind Eigentum der Unigraphics Solutions Inc. und urheberrechtlich geschützt. © 2002. Alle Rechte vorbehalten.

Teile dieser Software sind Eigentum der combit® GmbH und urheberrechtlich geschützt. Report- / Druckmodul List & Label® Version 8.0: Copyright combit® GmbH 1991-2004.

### **Änderungen**

Darüber hinaus behält sich DELMIA das Recht von Änderungen und Verbesserungen des in diesem Handbuch beschriebenen Produkts zu jeder Zeit und ohne Ankündigung vor.

DELMIA und das 3DS Logo sind eingetragene Warenzeichen von Dassault Systèmes oder Ihren Tochtergesellschaften in den Vereinigten Staaten oder in anderen Ländern.

Copyright © Dassault Systèmes 2001, 2007

# Inhaltsverzeichnis

<b>SKRIPTING</b>	<b>1</b>
<b>Vorwort</b>	<b>2</b>
<b>Inhaltsverzeichnis</b>	<b>4</b>
<b>Einleitung</b>	<b>11</b>
<b>Wie Sie Zeichen und Symbole lesen</b>	<b>12</b>
<b>1.1. Neue Funktionen im Skripting</b>	<b>13</b>
1.1.1. Neue und geänderte Funktionen in der Version PE 5.18	13
1.1.2. Neue und geänderte Funktionen in der Version PE 5.17	13
1.1.2.1. Erweiterte Klassen:	13
1.1.3. Neue und geänderte Funktionen in der Version PE 5.16SP4	14
1.1.3.1. Neue Funktionen	14
1.1.3.2. Erweiterte Klassen:	14
1.1.4. Neue und geänderte Funktionen in der Version PE 5.16	15
1.1.4.1. Neue Funktionen	15
1.1.5. Neue und geänderte Funktionen in der Version PE 5.15	16
1.1.5.1. Neue Funktionen	16
1.1.5.2. Geänderte Funktionen	16
1.1.5.3. Einschränkungen	17
1.1.6. Neue und geänderte Funktionen in der Version PE 5.14	17
1.1.6.1. Neue Funktionen	17
1.1.6.2. Geänderte Funktionen:	17
1.1.6.3. Einschränkungen:	18
1.1.7. Neue und geänderte Funktionen in der Version PE 5.13	18
1.1.7.1. Neue Funktionen	18
1.1.7.2. Erweiterte Klassen:	18
1.1.7.3. Geänderte Funktionen	19
1.1.8. Neue und geänderte Funktionen in der Version PE 5.12	19
1.1.8.1. Änderungen in der Version PE 5.12	20
1.1.9. Neue Funktionen in der Version PE 5.11	20
1.1.10. Neue Funktionen in der Version PE 5.10	21
1.1.10.1. Auswirkungen der Änderungen in der Version PE 5.10SP1	21
1.1.11. Änderungen in der Version PE 5.9	22
<b>1.2. Skripting im Process Engineer</b>	<b>24</b>
1.2.1. Skripting im Process Engineer	26
1.2.1.1. Systemvoraussetzungen:	26
1.2.1.2. Die Skripting Schnittstelle des Process Engineers.	26
<b>1.3. Skripte im DELMIA Process Engineer verwalten</b>	<b>29</b>
1.3.1. Skripte, VBA-Makros und Skriptaktionen	29
1.3.2. Wo befinden sich die Skripte, VBA-Makros und Skriptaktionen im PPR-Navigator?	30
1.3.2.1. Wie werden Skripte erstellt?	30
1.3.2.2. Beschreibung der Parameter	31
1.3.3. Rechte im Skripting	32
1.3.3.1. Als Skriptbesitzer ein Skript ausführen	32
1.3.4. Skripte in eigener Transaktion ausführen	34

<b>1.4. Skripte ausführen</b>	<b>35</b>
<b>1.4.1. Starten eines Skriptes über „Skript ausführen“</b>	<b>35</b>
<b>1.4.2. Starten eines Skriptes im interaktiven Modus</b>	<b>36</b>
1.4.2.1. Skripte automatisch starten mit Hilfe von Skriptaktionen	36
1.4.2.2. Skriptaktionen allgemein	40
<b>1.4.3. Skriptzuweisungen anlegen</b>	<b>41</b>
1.4.3.1. Die Methode valuation	44
1.4.3.2. Skript über das Kontextmenü ausführen	46
1.4.3.3. Skript im Eigenschaftsdialog eines Objektes ausführen	48
<b>1.4.4. Skripte automatisch starten</b>	<b>51</b>
1.4.4.1. Mit Skriptaktionen Skripte automatisch starten	51
<b>1.4.5. Skripte im Batch-Modus starten</b>	<b>51</b>
1.4.5.1. Batch-Aufruf mit Hilfe einer Datei	51
1.4.5.2. Batch-Aufruf mit Hilfe eines Objektes	52
<b>1.5. Wie müssen Skripte im Process Engineer aussehen?</b>	<b>54</b>
1.5.1.1. Fehlerbehandlung	55
1.5.1.2. Rekursionen	56
1.5.1.3. Weiterführende Beispiele	58
<b>1.5.2. Skripte in anderen Skripten verwenden</b>	<b>60</b>
<b>1.5.3. Protokollieren der Skriptaktivitäten</b>	<b>62</b>
<b>1.6. VBA-Host für die Programmierung im Process Engineer verwenden</b>	<b>63</b>
<b>1.6.1. Einleitung</b>	<b>63</b>
<b>1.6.2. VBA-Skripte erstellen</b>	<b>63</b>
1.6.2.1. VBA-Umgebung (IDE) öffnen	64
1.6.2.2. Ansicht VBA-Umgebung	64
<b>1.6.3. VBA-Makros ausführen</b>	<b>66</b>
<b>1.6.4. Skripte in VBA-Makros umwandeln</b>	<b>67</b>
1.6.4.1. VBA Objekt Application	69
<b>1.6.5. Fehlerbereinigung von VBA-Makros.</b>	<b>70</b>
<b>2. LISTE ALLER SKRIPTAKTIONEN</b>	<b>71</b>
<b>2.1. Skriptaktion Neues Kind</b>	<b>73</b>
<b>2.2. Skriptaktion Kind hinzufügen</b>	<b>74</b>
<b>2.3. Skriptaktion Attribute setzen</b>	<b>75</b>
<b>2.4. Skriptaktion Neu</b>	<b>76</b>
<b>2.5. Skriptaktion Kopieren</b>	<b>77</b>
<b>2.6. Skriptaktion Verknüpfen</b>	<b>78</b>
<b>2.7. Skriptaktion Verschieben</b>	<b>79</b>
<b>2.8. Skriptaktion Löschen</b>	<b>80</b>
<b>2.9. Skriptaktion Kind entfernen</b>	<b>81</b>
<b>2.10. Skriptaktion Init Properties</b>	<b>82</b>

2.11. Skriptaktion Change Properties	84
2.12. Skriptaktionen Neue Version	86
2.13. Skriptaktion Planungsstatus setzen	87
2.14. Skriptaktion OpenProject	88
2.15. Skriptaktion CloseProject	89
2.16. ScriptAction SelectProject	90
2.17. Skriptaktion StartBalancing	91
2.18. Skriptaktion SaveBalancing	92
2.19. Skriptaktion DropProcessToBalancing	93
2.20. Skriptaktion Drucken (Print)	94
2.21. Skriptaktion Listendruck (PrintList)	95
2.22. Skriptaktion OpenGraphic	96
<b>3. XSCRIPTINGHOST KLASSEN</b>	<b>97</b>
3.1. Klasse ScriptItemData	98
3.1.1. Liste der ScriptItemData Methoden	101
3.1.1.1. AddComponent	101
3.1.1.2. AttributeExists	102
3.1.1.3. CacheChildIds	102
3.1.1.4. ChangeRelationship	103
3.1.1.5. ChangeRelationshipEx	104
3.1.1.6. CommitTransaction	105
3.1.1.7. CopyAutoRelationUsageData	106
3.1.1.8. CopyComponent	106
3.1.1.9. CopyTimeAnalysis	107
3.1.1.10. CreateComponent	108
3.1.1.11. CreateRelationshipEx	109
3.1.1.12. CreateRelationshipWithOwner	110
3.1.1.13. CreateRootOrLibraryComponent	112
3.1.1.14. DeleteComponent	114
3.1.1.15. DisableChildrenListFilter	115
3.1.1.16. ExecuteServerMethod	116
3.1.1.17. ExecuteServerMethodEx	117
3.1.1.18. GetAttributebyId	118
3.1.1.19. GetAttributesbyId	118
3.1.1.20. GetLinkedObjectAttributebyId	119
3.1.1.21. GetLinkedObjectAttributesbyId	120
3.1.1.22. GetFirstChild	121
3.1.1.23. GetNextChild	122
3.1.1.24. GetChildrenCount	123
3.1.1.25. GetObjectGUIDbyObjectId	124
3.1.1.26. GetObjectIdbyObjectGUID	125

3.1.1.27.	GetOtherRelatedObject	126
3.1.1.28.	GetRelationshipsForOwnerByName	127
3.1.1.29.	IsDerivedFromClass	128
3.1.1.30.	IsDerivedFromType	129
3.1.1.31.	ResetIterator	130
3.1.1.32.	RemoveComponent	131
3.1.1.33.	RollbackTransaction	132
3.1.1.34.	SetOrderAttribute	133
3.1.1.35.	SetAttributebyId	134
3.1.1.36.	SetAttributeRange	135
3.1.1.37.	SetAttributesbyId	136
3.1.1.38.	SetFetchingSize	137
3.1.1.39.	SetLinkedObjectAttributesbyId	138
3.1.1.40.	SetLinkedObjectAttributebyId	139
3.1.1.41.	ReadSessionData	140
3.1.1.42.	WriteSessionData	141
<b>3.2.</b>	<b>Klasse SkriptItemRights</b>	<b>143</b>
3.2.1.	Liste aller XSkriptItemRights Methoden	144
3.2.1.1.	GetUserLogin	144
3.2.1.2.	GetUserFullName	144
3.2.1.3.	GetCurrentUser(PE 5.12)	145
3.2.1.4.	Transfer	146
3.2.1.5.	Transfer (PE 5.12)	147
3.2.1.6.	GetSingleRight	149
3.2.1.7.	GetSingleRight (PE 5.12)	150
3.2.1.8.	SetSingleRight (PE 5.12)	151
3.2.1.9.	GetAllGroups	152
3.2.1.10.	GetAllGroups (PE 5.12)	153
3.2.1.11.	GetAllUsers	153
3.2.1.12.	GetAllUsers (PE 5.12)	154
3.2.1.13.	GetUserInfoById	154
3.2.1.14.	GetGroupInfoById	155
3.2.1.15.	GetUserMemberships	155
3.2.1.16.	GetUserMemberships (PE 5.12)	156
3.2.1.17.	GetGroupMembers	157
3.2.1.18.	GetGroupMembers (PE 5.12)	158
3.2.1.19.	AddRights	160
3.2.1.20.	RemoveRights	161
3.2.1.21.	ConvertRightMaskToArray	162
3.2.1.22.	Create	163
3.2.1.23.	GetFunctionRights	164
3.2.1.24.	GetRightSubjects	165
<b>3.3.</b>	<b>Klasse SkriptItemGrid</b>	<b>166</b>
3.3.1.	Liste aller XSkriptItemGrid Methoden	166
3.3.1.1.	Create	166
3.3.1.2.	SetCell	167
<b>3.4.</b>	<b>Klasse SkriptItemGraphic</b>	<b>168</b>
3.4.1.	Liste aller XSkriptItemGraphic Methoden	169
3.4.1.1.	CreateBitmap	169
3.4.1.2.	CreateContextBitmap	170
3.4.1.3.	GetBBoxSize	171
3.4.1.4.	CreateFile	172

3.4.1.5.	CreatePreviewGraphic	173
3.4.1.6.	ShowGraphic	174
3.4.1.7.	GetGraphicMatrix	175
3.4.1.8.	GetGraphicRGB	176
3.4.1.9.	GetGraphicAlpha	177
3.4.1.10.	GetTranslation	177
3.4.1.11.	GetRotation	178
3.4.1.12.	SetGraphicMatrix	179
3.4.1.13.	SetGraphicRGB	180
3.4.1.14.	SetGraphicAlpha	180
3.4.1.15.	SetTranslation	181
3.4.1.16.	SetRotation	182
<b>3.5.</b>	<b>Klasse ScriptItemDialog</b>	<b>183</b>
3.5.1.	Liste aller XScriptItemDialog Methoden	184
3.5.1.1.	MessageBox	184
3.5.1.2.	MessageBoxExt	184
3.5.1.3.	CreateInputControl	185
3.5.1.4.	ModifyInputControl	188
3.5.1.5.	InputBox	189
3.5.1.6.	GetInputControlValue	190
3.5.1.7.	GetValueCB	191
3.5.1.8.	GetValueCKB	191
3.5.1.9.	GetValueEdit	191
3.5.1.10.	SetValueCB	192
3.5.1.11.	SetValueCKB	192
3.5.1.12.	SetValueEdit	192
3.5.1.13.	FileSelector (PE 5.7)	193
3.5.1.14.	FileSelector (PE 5.9)	194
3.5.1.15.	PropSetValues	196
3.5.1.16.	PropGetValues	197
3.5.1.17.	PropGetCurrentValue	198
3.5.1.18.	PropModifyAccess	199
3.5.1.19.	PropModifyBGColor	199
3.5.1.20.	PropModifyFGColor	200
3.5.1.21.	PropModifyFontSize	200
3.5.1.22.	PropModifyFontStyle	201
3.5.1.23.	PropModifyFontType	201
3.5.1.24.	PropModifyRep	202
3.5.1.25.	PropModifyStyle	203
3.5.1.26.	PropModifyVisibility	203
3.5.1.27.	Redraw	204
3.5.1.28.	RedrawAttribute	204
3.5.1.29.	RedrawGroup	205
3.5.1.30.	RedrawPage	205
<b>3.6.</b>	<b>Klasse ScriptItemList</b>	<b>206</b>
3.6.1.	Liste aller XscriptItemList Methoden	206
3.6.1.1.	CreateListView(PE 5.8)	206
3.6.1.2.	CreateListView(PE 5.9)	207
3.6.1.3.	OpenFinder	208
<b>3.7.</b>	<b>Klasse ScriptItemQuery</b>	<b>209</b>
3.7.1.	Liste aller ScriptItemQuery Methoden	210
3.7.1.1.	SetQuery	210



3.7.1.2.	SetConcatenator	213
3.7.1.3.	GetFirstResult	214
3.7.1.4.	GetNextResult	215
3.7.1.5.	GetResultCount	216
3.7.1.6.	GetOnlyFirstResult	217
3.7.1.7.	ResetSearch	217
3.7.1.8.	SetOrderAttribute	218
3.7.1.9.	BeginSubQuery	219
3.7.1.10.	EndSubQuery	220
3.7.1.11.	SetSubQuery	221
3.7.1.12.	SetQueryMode	222
3.7.1.13.	CacheResultIds	223
3.7.1.14.	SetFetchingSize	223
3.7.1.15.	SetOQLQuery	224
3.7.1.16.	IsObjectValid	224
3.7.1.17.	UseProjectFilter	225
3.7.1.18.	IsFilterActive	225
<b>3.8.</b>	<b>Klasse ScriptItemConfig</b>	<b>227</b>
3.8.1.	Liste aller XScriptItemConfig Methoden	227
3.8.1.1.	GetAllAttributes	227
3.8.1.2.	GetAllPlanTypes	228
3.8.1.3.	GetAllTypes	228
3.8.1.4.	GetTypeInfoTN	229
3.8.1.5.	GetTypeInfo	230
3.8.1.6.	GetAttributeInfoTN	231
3.8.1.7.	GetAttributeInfo	231
3.8.1.8.	GetAttributeValueList	232
3.8.1.9.	GetAttributeValueInfo	233
3.8.1.10.	GetParentPCRelations	233
3.8.1.11.	GetPCRelations	234
3.8.1.12.	GetPCRelationInfo	234
3.8.1.13.	ResetItem	235
<b>3.9.</b>	<b>Klasse ScriptItemConvert</b>	<b>236</b>
3.9.1.	Liste aller XscriptItemConvert Methoden	236
3.9.1.1.	Rtf2PlainText	236
3.9.1.2.	TranslateText	237
<b>3.10.</b>	<b>Klasse ScriptItemVersion</b>	<b>238</b>
3.10.1.	Liste aller XScriptItemVersion Methoden	239
3.10.1.1.	Create	239
3.10.1.2.	CreateDeep	240
3.10.1.3.	CheckOut	241
3.10.1.4.	CheckOutDeep	242
3.10.1.5.	CanPlanningStateBeModified	243
3.10.1.6.	SetPlanningState	244
3.10.1.7.	GetPlanningState	245
3.10.1.8.	GetPlanningStates	246
3.10.1.9.	GetFirstVersion	247
3.10.1.10.	CreateReleaseTable	248
3.10.1.11.	GetReleaseTable	249
3.10.1.12.	CreateSFI	249
3.10.1.13.	CreateDeepSFI	250
3.10.1.14.	CheckOutSFI	251

3.10.1.15. CheckOutDeepSFI	252
3.10.1.16. Promote	253
3.10.1.17. PDXMLFileCreated	253
3.10.1.18. Use	254
3.10.1.19. UpdateRelations	255
<b>3.11. Klasse ScriptItemUnit</b>	<b>256</b>
<b>3.11.1. Liste aller XScriptItemUnit Methoden</b>	<b>256</b>
3.11.1.1. Convert	256
3.11.1.2. GetDefaultUnitByAttr	257
3.11.1.3. GetDefaultUnitByCat	257
3.11.1.4. GetDefaultUnitByUnit	258
3.11.1.5. GetUIDefaultUnitByAttr	258
3.11.1.6. GetUIDefaultUnitByCat	259
3.11.1.7. GetUIDefaultUnitByUnit	259
3.11.1.8. GetAllUnitsByAttr	260
3.11.1.9. GetAllUnitsByCat	261
3.11.1.10. ConvertToDefaultUnit	262
3.11.1.11. ConvertToUIDefaultUnit	262
<b>3.12. Klasse ScriptItemLock</b>	<b>264</b>
<b>3.12.1. Liste aller XScriptItemLock Methoden</b>	<b>265</b>
3.12.1.1. LockObject	265
3.12.1.2. UnlockObject	266
3.12.1.3. UnlockAllObject	267
3.12.1.4. IsObjectLocked	268
3.12.1.5. GetObjectLockInfo	269
<b>INDEX</b>	<b>270</b>

## Einleitung

Moderne Betriebssysteme stellen Mittel und Wege bereit, um Routineaufgaben über Skripte zu automatisieren. Die Windows-Welt hat lange Zeit nur die Batch-Programmierung auf Kommandozeilenebene gekannt, diese Möglichkeiten waren jedoch sehr beschränkt. Mit der standardmäßigen Einführung eines leistungsfähigen Skript-Interpreters, dem *Windows Scripting Host* (WSH), wurde dieses Manko behoben.

Um die vielfältigen Möglichkeiten des Skripting im *Process Engineer* anzuwenden, können Sie die auf Ihrem System installierten Interpreter für Skriptsprachen benutzen. Im Allgemeinen wird dies von Systembibliotheken für *VBScript* (Visual Basic Script) und *JScript* (Java Script) geleistet, die Bestandteil von MS Windows und/oder MS Internet Explorer sind. Die Erfahrung zeigt, dass die meisten Benutzer Skripte in Basic schreiben, aus diesem Grund sind die im Handbuch gezeigten Beispiele in VBScript geschrieben.

Weitere Scriptsprachen, wie Perl, Tel, Python oder REXX können über entsprechende Sprach-Engines ebenfalls in den WSH integriert werden. Welche Sprache Sie einsetzen wollen, hängt davon ab, welche Vorkenntnisse Sie besitzen und welche Sprache Ihnen eher zusagt

### Wie setzen Sie nun dieses Handbuch ein?

- In erster Linie dürfen Skripte ausschließlich von einem Administrator oder einem gleichberechtigtem Mitarbeiter geschrieben werden. Dazu müssen diese Mitarbeiter neben den fachlichen Kenntnissen die Rechte besitzen, Skripte zu erstellen und auszuführen.
- Das Handbuch ist so aufgebaut, dass sowohl Einsteiger und Fortgeschrittene sich schnell in die Materie einfinden werden. Für den Einsteiger empfiehlt es sich, alle Kapitel zu lesen. Fortgeschrittene Anwender können die ersten Kapitel, die sich mit der Verwaltung von Skripten befassen, überspringen und gleich mit dem Kapitel Dokumentation der Xscripting-HostKlassen zu lesen beginnen.

Siehe auch: [XScriptingHost Klassen](#)

- In der Einleitung werden Neuheiten und Änderungen gegenüber Vorgängerversionen aufgezeigt. Dieser Teil sollte vor allem von dem fortgeschrittenen Anwender gelesen werden.



---

**Hinweis:** Denken Sie daran, zu den in diesem Handbuch beschriebenen Funktionen für das Skripting sollten Sie das Wissen aus dem Basis Handbuch hinzuziehen, in dem die allgemeine Einführung in den Process Engineer beschrieben wird.



Hier rufen Sie das Benutzer Handbuch [Basis Handbuch](#) auf.

---

## Wie Sie Zeichen und Symbole lesen

Die Zeichen und Symbole, die in diesem und in allen weiteren Handbüchern verwendet werden, dienen nicht nur zur allgemeinen Verschönerung eines Handbuchs, obwohl das auch eine der Aufgaben ist, sie dienen vor allem der Benutzerführung, um Ihnen den Inhalt auf leicht verständliche Weise zu erklären. Kapitel und Kapitelabschnitte werden durch Überschriften eingeleitet. Die Überschriften haben entsprechend der Verwendung unterschiedliche Schriftgrößen.

Nachfolgend wird Ihnen die Bedeutung der Symbole erklärt:



Mit diesem Symbol werden Textstellen bezeichnet, die den Funktionsumfang beschreiben, den Sie in einem Kapitel kennen lernen werden. Es steht daher in der Regel am Anfang eines Kapitels oder Abschnitts. Zudem werden wichtige Textstellen mit diesem Zeichen hervorgehoben.



**Hinweis:** Mit diesem Symbol werden Hinweise gekennzeichnet, die zu einem Thema noch zusätzliche Informationen liefern, die für das Weiterarbeiten sehr wichtig sind. Das Hinweis-Zeichen kann sowohl an einem Kapitelanfang als auch bei einer bestimmten Textstelle im Kapitel stehen. Die Texte, die mit diesem Zeichen eingeleitet werden, sind zusätzlich mit dem Wort Hinweis gekennzeichnet. Der Text selbst ist immer kursiv geschrieben.



**Achtung:** Mit diesem Zeichen werden Sie auf Sachverhalte aufmerksam gemacht, die zu möglichen Fehlern bei der Bedienung des Programms führen könnten und die Sie daher beachten sollten. Das Achtung-Zeichen kann sowohl an einem Kapitelanfang als auch bei einer bestimmten Textstelle im Kapitel stehen. Die Texte, die mit diesem Zeichen eingeleitet werden, sind zusätzlich mit dem Wort Achtung gekennzeichnet. Der Text selbst ist immer kursiv geschrieben.

### Beispiel

Mit diesem Symbol werden Sie auf Beispiele aufmerksam gemacht, die einen Sachverhalt verdeutlichen.

- Mit diesem Symbol werden die einzelnen Bedienschritte einer Handlungsanweisung gekennzeichnet. Mit Handlungsanweisungen werden Bedienschritte beschrieben, um beispielsweise ein Menü zu öffnen oder eine Funktion auszuführen.
- Mit diesem Symbol werden Aufzählungen gekennzeichnet. Das Aufzählungssymbol kann sowohl für eine Gliederung eines Fließtextes verwendet werden als auch stichpunktartig Themenschwerpunkte aufzulisten.



Mit diesem Symbol werden Sie darauf aufmerksam gemacht, dass es zu diesem Thema noch weitere Informationen in einem anderen Handbuch gibt.

## 1.1. Neue Funktionen im Skripting



### Hinweis

---

*Dieses Handbuch führt Sie in die Basisfunktionen des Skriptings im DPE ein. Um den aktuellsten Stand des Skriptings für jedes Release zu erhalten, lesen Sie zudem bitte auch das englische Benutzerhandbuch Skripting.*

---

### 1.1.1. Neue und geänderte Funktionen in der Version PE 5.18

### 1.1.2. Neue und geänderte Funktionen in der Version PE 5.17

#### 1.1.2.1. **Erweiterte Klassen:**

- ☐ Erweiterte Funktionen der Skriptzuweisungen (siehe auch: [Skriptzuweisungen anlegen](#)).
  - Ab der Version PE 5.17 ist es möglich Skriptzuerisungen für einzelne Seiten, Gruppen und Attribute zu definieren.
- ☐ Neue Funktion in Data [CreateRelationshipEx](#)
- ☐ Neue Funktion in Version [UpdateRelations](#).
- ☐ Verbunden mit den Skriptaktionen [Skriptaktion Change Properties](#) und [Skriptaktion Init Properties](#)
  - [PropModifyVisibility](#) (Versteckt/Zeigt veränderte Layout Objekte)
  - [PropModifyAccess](#) (Sperrt/Entsperrt den Zugang für Attribute)
  - [PropModifyBGColor](#) (Ändert die Hintergrundfarbe)
  - [PropModifyFGColor](#)(Ändert die Vordergrundfarbe)
  - [PropModifyFontSize](#) (Ändert die Schriftfarbe)
  - [PropModifyFontStyle](#) (Ändert den Schriftstil)
  - [PropModifyFontType](#) (Ändert die Schriftart)
  - [PropModifyRep](#)
  - [Redraw](#) (Neuer Entwurf aller Seiten)
  - [RedrawPage](#) (Neuer Entwurf einer Seite)
  - [RedrawGroup](#) (Neuer Entwurf einer Gruppe)
  - [RedrawAttribute](#) (Neuer Entwurf eines Attributes)
- ☐ Skriptingmethoden für den Zugang zum OpenProject Dialog: Verbunden mit der neuen Skriptaktion [ScriptAction SelectProject](#)

- [GetValueCB](#) (Comboboxwert erhalten)
- [GetValueEdit](#) (Änderungswert erhalten)
- [GetValueCKB](#) (Checkboxwert erhalten)
- [SetValueCB](#) (Comboboxwert einstellen)
- [SetValueEdit](#) (Änderungswert einstellen)
- [SetValueCKB](#) (Checkboxwert einstellen)

### 1.1.3. Neue und geänderte Funktionen in der Version PE 5.16SP4

#### 1.1.3.1. **Neue Funktionen**

- ❑ In der Version PE 5.16SP4 wurden folgende Funktionen hinzugefügt:
  - [Skriptaktion Listendruck \(PrintList\)](#)
- ❑ Neue Funktionen in Data [ExecuteServerMethodEx](#)
- ❑ Neue Funktionen in Grafik [ShowGraphic](#)
- ❑ Neue Funktion in Rechte [GetRightSubjects](#)
- ❑ Neue Funktionen in Version [Promote](#)
- ❑ Neue Funktionen in Version [CanPlanningStateBeModified](#)

#### 1.1.3.2. **Erweiterte Klassen:**

- ❑ Erweiterte Funktionen der Skriptzuweisungen (siehe auch: [Skriptzuweisungen anlegen](#)).
  - Jetzt ist es möglich zu differenzieren ob ein verknüpftes Skrip für jedes einzelne Objekt einer mehrfach Auswahl ausgeführt werden muss oder für alle Objekte.
  - Ob ein Eintrag in das Kontextmenü erzeugt werden soll.
  - Ob in den verknüpften Objekten ein Button für Ausführung erzeugt werden soll
- ❑ [Skriptaktion Change Properties](#) (sa\_changeproperties): Bisher war die Initialisieren des Eigenschaften Dialogs auf die List- und Combobox beschränkt. Ab der Version 5.16SP4 wurde diese Skriptaktion auf Attribute mit dem Controltyp:
  - Edit
  - Multi Line Edit
  - Check Box
  - Radio Buttons
  - Date Controlerweitert.

## 1.1.4. Neue und geänderte Funktionen in der Version PE 5.16

### 1.1.4.1. Neue Funktionen

- ❑ In der Version PE 5.16 wurden folgende Funktionen hinzugefügt:
  - [CreatePreviewGraphic](#) (Erzeugt Grafikdateien von DPE Grafiken, die in DPM angesehen werden können)
  - [AttributeExists](#) (Überprüft das Vorliegen eines Attributes von einem Objekt.)
  - [CreateRelationshipEx](#) (Erzeugt eine Verknüpfung zwischen Objekten – mit oder ohne Besitzer)
  - [ChangeRelationship](#) (ändert verknüpfte Objekte einer Relation)
  - [UnlockAllObject](#) (um alle gelockten Objekte auf einmal freigeben)
  - Infolge der *shop floor integration (SFI)* wurden einige neue Funktionen in PE5.16 eingeführt. Bezüglich der Bedienung von ReleaseTables gibt es neue Eigenschaften:
    - [CreateReleaseTable](#) (die erste Erstellung einer Versions- Tabelle -> ReleaseTables)
    - [GetReleaseTable](#) (Zugriff auf eine existierenden ReleaseTables)
    - [PDXMLFileCreated](#) (Erstellt einen Zeigers auf einen ReleaseTable – Eintrag, der von einer pdxml – Datei erstellt wurde.)
  - Die vorhandenen Methoden:
    - [Create](#)
    - [CheckOut](#)
    - [CreateDeep](#)
    - [CheckOutDeep](#) berücksichtigt die neu eingeführten SFI Konsistenzprüfungen. Sie können von den neuen Methoden überschrieben werden.
    - [CreateSFI](#)
    - [CreateDeepSFI](#)
    - [CheckOutSFI](#)
    - [CheckOutDeepSFI](#)
- ❑ Einführung des zusätzlichen Scriptsaction *Delete (After)* in PE5.15 und später.
- ❑ Rekursives Holen von Kindern (application server side change) ist in PE5.15 und später möglich.

## 1.1.5. Neue und geänderte Funktionen in der Version PE 5.15

### 1.1.5.1. Neue Funktionen

- ❑ In der Version PE 5.15 wurden folgende Funktionen hinzugefügt:
  - [IsDerivedFromType](#) (ersetzt Data.IsDerivedFromClass)
  - [AttributeExists](#) (Überprüft das Vorliegen eines Attributes von einem bestimmten Objekt) in PE5.15SP3 und später.
  - CompLock.UnlockAllObjects (um alle gelockten Objekte auf einmal freizugeben), in PE5.15SP3 und später.
  - Einführung der zusätzlichen Skriptaktion [Skriptaktion Löschen \(After\)](#) in PE5.15SP3 und später.
- ❑ Rekursives Holen von Kindern (application server side change) ist in PE5.15SP3 und später möglich.

### 1.1.5.2. Geänderte Funktionen

Aus funktionaler Sicht gibt es fast keine Unterschiede zu der Version PE5.14, jedoch sind einige wichtige Veränderungen, durch die Einführung von Single Sign-On (SSO) und die Entschlüsselung der Passwörter zwingend notwendig geworden.

- ❑ Die Kommandozeile für die Batch Ausführung muss geändert werden. Siehe auch [Skripte im Batch-Modus starten](#)
- ❑ Beim Erzeugen eines neuen Benutzers muss das Passwort, vor der Einstellung für das Passwortattribut, verschlüsselt werden. Siehe auch [Create](#).
- ❑ In Query's wird nicht mehr der physikalische Namen (z. B. "[XDOErgo-CompProcessdefault](#)", "[m\\_pErgoProject](#)", etc.) als Parameter in den Funktionen des Abfrageobjektes verwendet. Stattdessen müssen Sie den konfigurierte Namen (z. B. "[ergocompprocessdefault](#)", "[ergoproject](#)", etc.) verwenden
- ❑ Die Klasse ***XDOSubCompltem*** ist jetzt von ***XDORelationship*** abgeleitet. Das erfordert einige Veränderungen, besonders für die Abfragen (Queries). Die Klasse ScriptItemData ist ebenfalls davon betroffen, aber einige der Aufrufe werden automatisch im Server noch mit den alten Aufrufen, um die Abwärtskompatibilität zu gewährleisten, ausgeführt.
  - Data.GetChildren(*parent\_id*, "[subcompitem](#)") wird nicht mehr unterstützt.  
-> Benützen Sie stattdessen Data.GetChildren(*parent\_id*, "[nodes](#)") .
  - Eine Abfrage auf "[XDOSubCompltem](#)" ist nicht mehr möglich. Stattdessen ist der korrekte tablename "[relationship\\_nodes](#)". Zudem ersetzen Sie die Attributnamen "[m\\_pDODefaultImpl](#)" oder "[m\\_pErgoCompBaseParent](#)" mit "[relationobjectsource](#)" und "[m\\_pErgoCompBase](#)" mit "[relationobjecttarget](#)".
  - Data.GetAttributebyId(*sci\_id*, "[ergocompbase](#)") funktioniert nach wie vor, wird aber mit "[relationobject2](#)" im Server aufgerufen. Es wird deshalb empfohlen Data.GetAttributebyId(*sci\_id*, "[relationobject2](#)") direkt aufzurufen.



"relationobject2" ist mit "relationobjecttarget" in der Abfrage äquivalent.\*

- Data.GetAttributebyId(sci\_id, "ergocompbase\_parent") funktioniert nach wie vor, wird aber mit "relationobject1" im Server aufgerufen. Es wird deshalb empfohlen *Data.GetAttributebyId(sci\_id, "relationobject1")* direkt aufzurufen.  
"relationobject1" ist mit "relationobjectsource" in der Abfrage äquivalent.\*



**Hinweis:** Die Beispielskripte in dieser Dokumentation benutzen immer noch "ergocompbase" und "ergocompbase\_parent".

- ☐ Im Falle, dass die Skriptaktion abgeschaltet ist und die Anmeldung freigegeben ist, wird kein Logeintrag mehr eingetragen.
- ☐ Wenn Sie die Klasse ScriptItemQuery verwenden, dann überprüfen Sie den Namen der Querytable für jede Abfrage (die Hauptabfrage und jede Unterfrage), selbst wenn sich der Name für die nächsten Funktionsaufruf nicht ändern sollte.

### 1.1.5.3. Einschränkungen

- ☐ Die "Change Properties" Skriptaktion ist zur Zeit auf Comboboxen und Listboxen begrenzt.

## 1.1.6. Neue und geänderte Funktionen in der Version PE 5.14

### 1.1.6.1. Neue Funktionen

Der Batch-Aufruf für Skripte wurde um eine Dateibezogene Stapelausführung von Skripten erweitert (zusätzlich zu den Skriptobjekt Stapelausführungen).

#### ☐ Erweiterte Klassen:

- Data.DisableChildrenListFilter (um ungefilterte Kinderlisten zu erhalten)
- Data.SetAttributeRange (um eine Liste von Objekten zu erhalten, die von einem eindeutigen Planungstyp innerhalb eines vordefinierten Attribut-Wertebereich instanziiert wurden.)
- Data.ReadSessionData (um Lesezugriff auf die Registrierungseditor-Daten zu erhalten, ähnlich wie „Werkzeuge/Einstellungen/Wartung...“).
- Data.WriteSessionData (um Schreibzugriff auf die Registrierungseditor-Daten zu erhalten).

#### ☐ Eine zusätzlich Skriptaktion wurde eingeführt:

- Öffne Grafik (before)

### 1.1.6.2. Geänderte Funktionen:

- ☐ Die (Pseudo-) Skriptaktionen wurden durch **Skriptzuweisungen** ersetzt.
- ☐ Aufgrund einer Änderung des Datenmodells besitzen Objekte des Typs

- „subcompitem“ (Stücklisteneinträge, BOM Einträge) und
- „subcompviewitem“ (beinhalten BOM views)

keine Zeiger (Pointer) auf "m\_pBom" (Konfigurationsname "bom") mehr.

Skripte, die ausdrücklichen Gebrauch von dem Attribut „bom“ beim Aufruf von `Data.GetAttributebyId` machen oder das Attribut "m\_pBom" bei Suchanfragen verwenden, müssen geändert werden.

Der Funktionsaufruf:

- `bom_id = Data.GetAttributebyId(sci_id, "bom")`  
*ist nicht mehr gültig.*
- Auf der anderen Seite bleibt die Ausführung  
`parent_ecb_id = Data.GetAttributebyId(sci_id, "ergocompbase_parent")` nicht nur gültig, sondern besitzt, durch eine speziellen Server Implementation, einen physikalischen Link im Datenmodell. Mit anderen Worten referenziert jetzt, ab PE 5.14 und später, ein Stücklisteneintrag seine Basiskomponente direkt.

Bei Suchanfragen ersetzen Sie bitte den Funktionsaufruf:

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

- durch

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

Beachten Sie, dass die Stücklisteneinträge weiteren Änderungen in späteren Versionen unterzogen werden könnten. Es ist wahrscheinlich, dass Stücklisteneinträge in der Zukunft durch spezielle Verknüpfungen ersetzt wird.

#### 1.1.6.3. **Einschränkungen:**

Die Skriptaktion „Change Properties“ ist auf Combobox und Listbox Controls beschränkt.

### 1.1.7. Neue und geänderte Funktionen in der Version PE 5.13

#### 1.1.7.1. **Neue Funktionen**

- ☐ **Zwei zusätzlichen Skriptaktionen in Bezug auf den Eigenschaften Dialog**

- Initialisieren des Eigenschaften Dialogs
- Wechseln der Eigenschaften

Damit wurden auch neue Funktionen in der Klasse `XscriptItemDialog` notwendig.

#### 1.1.7.2. **Erweiterte Klassen:**

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

Mit diesen neuen Funktionen können Abhängigkeiten zwischen Attributen in einem Eigenschaftsdialog definiert werden.

- CreateRootOrLibraryComponent

#### ☐ Erzeugen von Objekten an Wurzelobjekten

Ab der Version PE 5.13 können Sie mit Skriptfunktionen Kinder an Wurzelobjekten hinzufügen. Dies sind im Einzelnen:

1. ProjectRoot (Projekte erzeugen)
2. ArchivRoot (Objekte in der globalen Bibliothek erzeugen)
  - Data.CreateRootOrLibraryComponent
3. RightsRoot (Benutzer und Gruppen erzeugen)
  - XscriptItemRights. Create
  - XscriptItemRights. GetFunctionRights

#### ☐ Starten des Suchers mit Hilfe von Skriptfunktionen

Mit dieser Skriptfunktion können Sie den *Projektsucher* mit vordefinierten Einstellungen starten und initialisieren.

- XscriptItemList. OpenFinder

### 1.1.7.3. Geänderte Funktionen

Am Typ *XDORelationship* ist das Attribut

- „m\_pOwnerErgoComponent“ durch „m\_pOwner“ ersetzt worden
- "owner" wurde in „relationowner“ umbenannt.

Wenn Sie „m\_pOwnerErgoComponent“ und „ownerergocomponent“ in einem Skript verwendet haben, müssen Sie diese in "m\_pOwner" beziehungsweise "relationowner" ändern.

#### ☐ Einschränkungen

- Die neue Skriptaktion „Wechseln der Eigenschaften“ ist gegenwärtig auf die Typen des Controls Combobox und Listbox beschränkt.
- Queries können nicht auf benutzerdefinierte Attribute angewendet werden.

## 1.1.8. Neue und geänderte Funktionen in der Version PE 5.12

#### ☐ Die Ländereinstellungen des Servers beim Ausführen von Skripten verwenden

Verwende Ländereinstellungen des Servers bei Ausführung von Skripten ☒

In Multiserverumgebungen mit verschiedenen lokalen Ländereinstellungen auf der Client- bzw. Serverseite können Sie nun Skripte mit Hilfe der lokalen Einstellungen des Servers ausführen, da sonst das Setzen bestimmter Objektattribute, die abhängig von den Ländereinstellungen sind, nicht funktioniert. Diese Einstellung nehmen Sie im Reiter *Scripting* der Einstellungen vor. Die *Einstellungen* finden Sie unter *Werkzeuge / Einstellungen / Ändern....* Damit werden die lokalen Einstellungen des Clients für die Laufzeit eines Skripts vorläufig auf die Einstellungen des Servers gestellt. Es wird empfohlen, diese Option nur dann zu aktivieren, wenn tatsächlich unterschiedliche Ländereinstellungen vorliegen.

### ☐ Ausführen eines Skriptes bei Mehrfachselektion

Sind mehrere Listeneinträge selektieren und Sie starten ein Skript, wird dieses Skript auf jedem der selektierten Einträge ausgeführt.

#### 1.1.8.1. Änderungen in der Version PE 5.12



**Hinweis:** Die folgenden Änderungen im Bereich der Serverinfrastruktur erforderten eine Anpassung vorhandener Skripte.

### ☐ Die Rechtsdatenbank ist ab der Version 5.12 Teil der Objektdatenbank.

Als eine Folge mussten viele der Funktionen und Methoden für das IScriptItemRights Objekt geändert werden. Skripte, die diese Objekte benutzen, müssen überarbeitet werden.

Gruppen-, Benutzer-, Objekt- und- Planungstypenrechte werden jetzt wie andere Objekte behandelt. Dies bedeutet besonders, dass das „Rechte“-Objekt sich wie alle anderen Objekte verhält. Ein Rechtsobjekt besitzt nun eine Objekt ID, und sobald Sie diese Objekt ID erhalten haben, können Sie nach dessen Attributen fragen. Die frühere Benutzer- und Gruppen- ID sind im Attribut „**Rightsobjectid**“ (aus Verträglichkeitsgründen) gespeichert.

### ☐ Vermeiden Sie den Gebrauch von internen Attributnamen bei Get-/SetAttribute (S) Byld Aufrufen.

#### Beispiel:

Obwohl das Attribut m\_pBom existiert, ist es nicht möglich, es innerhalb der Funktionen Get- / SetAttribute (S) Byld zu verwenden. Aus Leistungsgründen ist die Durchführung der Server Get- /SetAttribute- Funktionen völlig auf Kleinbuchstabenschreibung umgestellt worden. Im Server wird "m\_pBom" in m\_pbom umgewandelt und nicht mehr erkannt. Wird anstelle von "m\_pBom" die Attributbezeichnung „bom“ verwendet, arbeitet das Skript fehlerlos.

### ☐ Richtungsangabe bei Relationen

Wenn Sie mittels Skripten auf Relationen zugreifen, muss ab der Version 5.12 die Richtung der Relation berücksichtigt werden.

#### Beispiel:

Sie rufen auf einem **Produkt** die Funktionen *Data.AddComponent* oder *Data.GetFirst/NextChild* auf:

#### Alter Aufruf:

```
rel_id = Data.GetFirstChild(product_id,"proc_firstprocesses_prod")
```

#### Neuer Aufruf

```
relation_id = Data.GetFirstChild( product_id, "proc_firstprocesses_prod_reverse" )
```

Rufen Sie auf einem **Prozess** die gleiche Funktion und Relation auf, bleibt der Aufruf wie in den vorherigen Versionen.

## 1.1.9. Neue Funktionen in der Version PE 5.11

### ☐ Drei neue Skriptaktionen für die Austaktungsmodule.

- Start der Austaktung
- Speichern der Austaktung

- Prozesse in die Austaktungsliste einfügen
- ☐ XScriptItemData.CopyAutoRelationUsageData

### 1.1.10. Neue Funktionen in der Version PE 5.10

- ☐ Projekt unabhängige Skripte und Skriptaktionen
- ☐ Hinzufügen von Skripten in andere Skripte
- ☐ Skripte in eigener Transaktionen
- ☐ Die VBA Integration
- ☐ Neue Klassen sind:
  - XScriptItemConfig (der Zugriff auf die Konfigurations- Datenbank)
  - XScriptItemUnit (der Zugriff auf die Einheiten)
  - XScriptItemLock (Lock/Unlock Objekte mittels Skript)
  - XScriptItemConvert (Umwandlungs- Funktionen)

#### Geänderte (erweiterte) Klassen sich:

- ☐ XScriptItemRights (Lesen und Setzen von Rechten an Objekten, Typen und Funktionen)
- ☐ XScriptItemData (Umwandlung der Objekt-ID zu Objekt GUID)
- ☐ XScriptItemQuery (Zugriff auf gefilterte Datenbankabfragen, ausführen von OQL Datenbankabfragen)

#### 1.1.10.1. Auswirkungen der Änderungen in der Version PE 5.10SP1

##### ☐ GetFirstChild/GetNextChild

Aufgrund der Notwendigkeit, Änderungen am *Scripting Host* vorzunehmen, **müssen Sie vorhandene Skripte überarbeiten.**

Bisher war mit Hilfe der ID einer Subcompitem der Aufruf FirstChild / GetNextChild möglich. Subcompitem verweisen auf Objekte vom Typ ergocompbase, denen Kinderlisten zugehörig sind. Der *Scripting Host* hat den Funktionsaufruf von einem Subcompitem implizit auf den Typ ergocompbase umgeleitet.

Nach einem Upgrade auf R10 treten Probleme mit vorhandenen Skripten normalerweise in **verschachtelten** oder **rekursiven** BOM Strukturen auf, weil die zurückgegebenen Kinder-IDs *subcompitem* IDs sind.

```
REM parent_sci_id: ID der subcompitem
REM parent_id: ID des referenzierten ergocompbase-Objektes

REM FALSCHER Code für PE 5.10 und später
child_id = Data.GetFirstChild(parent_sci_id, child_listname)
Do while child_id <> ""
  child_id = Data.GetNextChild(parent_sci_id, child_listname)
Loop

REM RICHTIGE Code für PE 5.10 und später
parent_id = Data.GetAttributebyId(parent_sci_id, "ergocompbase")
child_id = Data.GetFirstChild(parent_id, child_listname)
Do while child_id <> ""
  child_id = Data.GetNextChild(parent_id, child_listname)
Loop
```

```

REM *****
REM ***Rekursives Lesen von Kindern ***
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 Die Rekursion; Zeigt den Namen der gefundenen Kinder
      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, "relationobject2")
end function

```

#### ❑ Der Ordner „Skriptaktionen“ im Projektplantypensatz (vorher Projektbibliothek)

Mit der Einführung der Plantypensatzpools in der Systembibliothek sind Skripte und Skriptaktionen nicht mehr von einem Projekt abhängig.

Durch das Verschieben der Skriptaktionen aus der Projektbibliothek in den Projektplantypensatz ändert sich nichts an der Funktionsweise der Skriptaktionen.

Im Allgemeinen sind Skriptaktionen nicht mehr einem Projekt zugehörig, sondern sind vom Planungstypensatz abhängig. Jede Skriptaktion kann mit einem beliebigen Planungstyp des Planungstypensatzes verbunden werden, zu dem die Skriptaktion gehört – man könnte auch sagen, Plantypen und Skriptaktionen sind Geschwister und die Kinder eines Plantypensatzes (das Elternteil). Daraus folgt, dass projektunabhängige Skriptaktionen zu einem der Planungstypensätze der Systembibliothek gehören.

Aus Gründen der Übersichtlichkeit und Vollständigkeit werden auch die Änderungen der Version PE 5.9 aufgezählt:

### 1.1.11. Änderungen in der Version PE 5.9

Betroffen sind: [SetAttributebyId](#) / [SetAttributesbyId](#)

[/SetLinkedObjectAttributebyId](#) / [/SetLinkedObjectAttributesbyId](#)

Ein einmal gesetztes Attribut einer **subcompitem** Komponente verweist nicht mehr auf die **ergocompbase** dieser Komponente. Dies hat zur Folge, dass Sie sich vor einem **set**-Aufruf einer subcompitem Komponente das Attribut der dazugehörigen ergocompbase Komponente erfragen müssen.

[Beispiel](#)

- ➔ Die Funktion **call** in der PE 5.7 und 5.8 setzt das Vorhandensein eines Basisobjektes voraus (base\_id).

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

- ➔ Die Funktion **call** muss ab der Version PE 5.9 wie nachfolgend dargestellt aufgerufen werden:

```
base_id = Data.GetAttributebyId(sci_id, "relationobject2")  
call Data.SetAttributebyId(base_id, "name", "New Name")
```

## 1.2. Skripting im Process Engineer

### Was ist Skripting?

Skripting bedeutet genau genommen nichts anderes als Programmieren! Dabei ist die wesentliche Aufgabe von Scripting, es Ihnen zu ermöglichen, den Funktionsumfang eines Betriebssystems und von Programmen auf spezifische Weise zu erweitern. Das gilt in erster Linie für kleinere Aufgaben, ist aber in der Komplexität nur durch die eingesetzte Skriptsprache und die angebotenen Möglichkeiten der Interaktion mit Programmen und Betriebssystem eingeschränkt. Skripting hilft Ihnen also dabei, Ihre Anwendungen flexibel zu steuern und Routinearbeiten zu automatisieren.

### Was ist der Windows Scripting Host?

Der **Windows Scripting Host** (kurz: = **WSH**) ist dafür zuständig, von Ihnen geschriebene Skripte auszuführen, d.h. er interpretiert das in einer bestimmten Sprache geschriebene Skript und gibt die entsprechenden Anweisungen an das Betriebssystem und die verwendeten Applikationen und Objekten weiter. Der WSH prüft dabei zunächst, ob das Skript syntaktisch korrekt ist, d.h. bezüglich der verwendeten Sprache keine „Rechtschreibfehler“ oder logische Fehler enthält und führt dann anschließend den Code aus.

### Womit schreibe ich Skripte?

In welchem Editor Sie ein Skript schreiben, ist ohne Belang. Kurze Skripte kann man mit einem einfachen Texteditor wie etwa mit *Notepad* oder *Wordpad* schreiben. Bei längeren Skripten empfiehlt sich der Einsatz eines Editors, der sog. *Syntax highlighting* erlaubt, d.h. sprachspezifische Elemente werden strukturiert (farbig/fett/blockweise) hervorgehoben und erleichtern die Übersicht.

### Was muss ich tun, um ein Skript auszuführen?

Im Allgemeinen genügt es, mit einem Texteditor ein Skript zu schreiben und diese Datei mit einer für den WSH erkennbaren Endung zu speichern. Standardmäßig sind dies \*.vbs für VBScript und \*.js für JScript. Der WSH geht also davon aus, dass in einer Datei mit der Endung \*.vbs ein Skript steckt, das in VBScript geschrieben ist und versucht dieses auszuführen. Auf diese sehr einfache Weise können Sie allerdings nicht mit Objekten des *Process Engineer* kommunizieren. Aus Gründen, auf die wir später eingehen werden, können Skripte, die mit dem *PPR Hub* kommunizieren, nur innerhalb des *Process Engineer* ausgeführt werden. Diese Einschränkung wurde also ganz bewusst eingebaut.

Öffnen Sie einen der Texteditoren *Notepad* oder *Wordpad* und tragen Sie folgenden Einzeiler ein:

```
MsgBox "Hallo, dies ist eine Ausgabebox", , "Mein erstes Skript"
```

- ➔ Speichern Sie nun diesen Eintrag als *MyScript.vbs* ab.

- ⇒ Danach sollte das Datei-Icon so aussehen:



- ➔ Führen Sie das Skript aus, indem Sie einen Doppelklick auf die Datei machen.



⇒ Als Ergebnis sollte folgendes herauskommen:



**Abbildung 1: Beispiel eines Skripts**

Die Methode *MsgBox* ist Bestandteil der Programmiersprache VBScript. Wie jede andere Programmiersprache verfügt VBScript über eine ganze Anzahl von *Funktionen* und *Kontrollstrukturen* (Schleifen, Verzweigungen), die Daten verarbeiten, die in Form von *Konstanten* und *Variablen* übergeben werden. Richtig mächtig wird ein Skript aber erst durch die Kommunikation mit *Objekten*, die vom Betriebssystem und anderen Anwendungen bereitgestellt werden. Eines dieser Objekte ist beispielsweise das *FileSystemObject*, das VBScript erlaubt, mit dem Dateisystem zu kommunizieren, insbesondere Dateien zu lesen und zu schreiben. Der folgende Quelltext erzeugt eine Datei namens `c:\test.txt`, die die Zeile „Diese Textdatei wurde per Skript generiert.“ ausgibt.

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



Der Process Engineer stellt spezielle Objekte bereit, die dem Benutzer erlauben, per Skript mit der Applikation zu interagieren (siehe auch Handbuch [Administration](#)).

### **Sicherheitsbedenken bei Einsatz von Skripten?**

Diskussionen um Computerviren wie *Melissa* oder *Code Red* sind allgegenwärtig und haben Skripte und Skriptinterpreter in Misskredit gebracht, weil diese Viren nichts anderes als Skripte sind, die zur Sabotage missbraucht werden. Die einfachste Lösung besteht darin, einfach alle Interpreter abzuschalten. Ist dies bei Ihnen der Fall, so können Sie sich das Weiterlesen ersparen, denn dann können Sie einfach *keine* Skripte ausführen. Auch ein Großteil von Internet/Intranet-Seiten wird nicht oder nicht mehr korrekt angezeigt werden. Es gibt, wie bei richtigen Viren auch, keinen 100% Schutz gegen Computerviren- außer Sie schotten sich rigoros von der Umwelt ab. Dies entspricht dem Abschalten der Skriptinterpreter in der Computerwelt. Wenn Sie sich vor (Skript-)Viren schützen und trotzdem nicht isoliert sein wollen, können wir Ihnen nur empfehlen, genau darauf zu achten, dass Sie nur Skripte ausführen, die Sie selbst geschrieben oder vorher angeschaut haben oder die Ihnen von vertrauenswürdiger Seite zur Verfügung gestellt wurden.

## 1.2.1. Skripting im Process Engineer

### 1.2.1.1. Systemvoraussetzungen:

Die Systemvoraussetzungen sind nicht besonders hoch:

#### **Windows NT 4**

Service Pack 3 und höher

Internet Explorer 4.01 oder höher

Windows® Scripting Host (WSH) muss installiert und aktiviert sein

#### **Windows2000® und Windows XP®**

Keine zusätzlichen Anforderungen

#### **DELMIA Process Engineer**

E5.7 und höher



---

**Hinweis:** Wenn im weiteren Verlauf des Textes vom Öffnen oder vom Zugriff auf andere Anwendungen als dem DELMIA Process Engineer die Rede ist, wird davon ausgegangen, dass diese Tools auch auf dem jeweiligen System vorhanden sind. Werden z. B. Daten nach MS-Excel transportiert, muss natürlich MS-Excel auf dem System installiert sein, damit das Skript ordnungsgemäß ausgeführt werden kann.

---

### 1.2.1.2. Die Skripting Schnittstelle des Process Engineers.

Der Process Engineer stellt eine eigene Schnittstelle zur Verfügung, die eine gewisse Untermenge aus den Zugriffsmethoden auf den PPR-Hub für das Skripting zugänglich macht. Die einzelnen Klassen sind:

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

Die genaue Beschreibung der Klassen finden Sie im Kapitel [XScriptingHost Klassen](#).



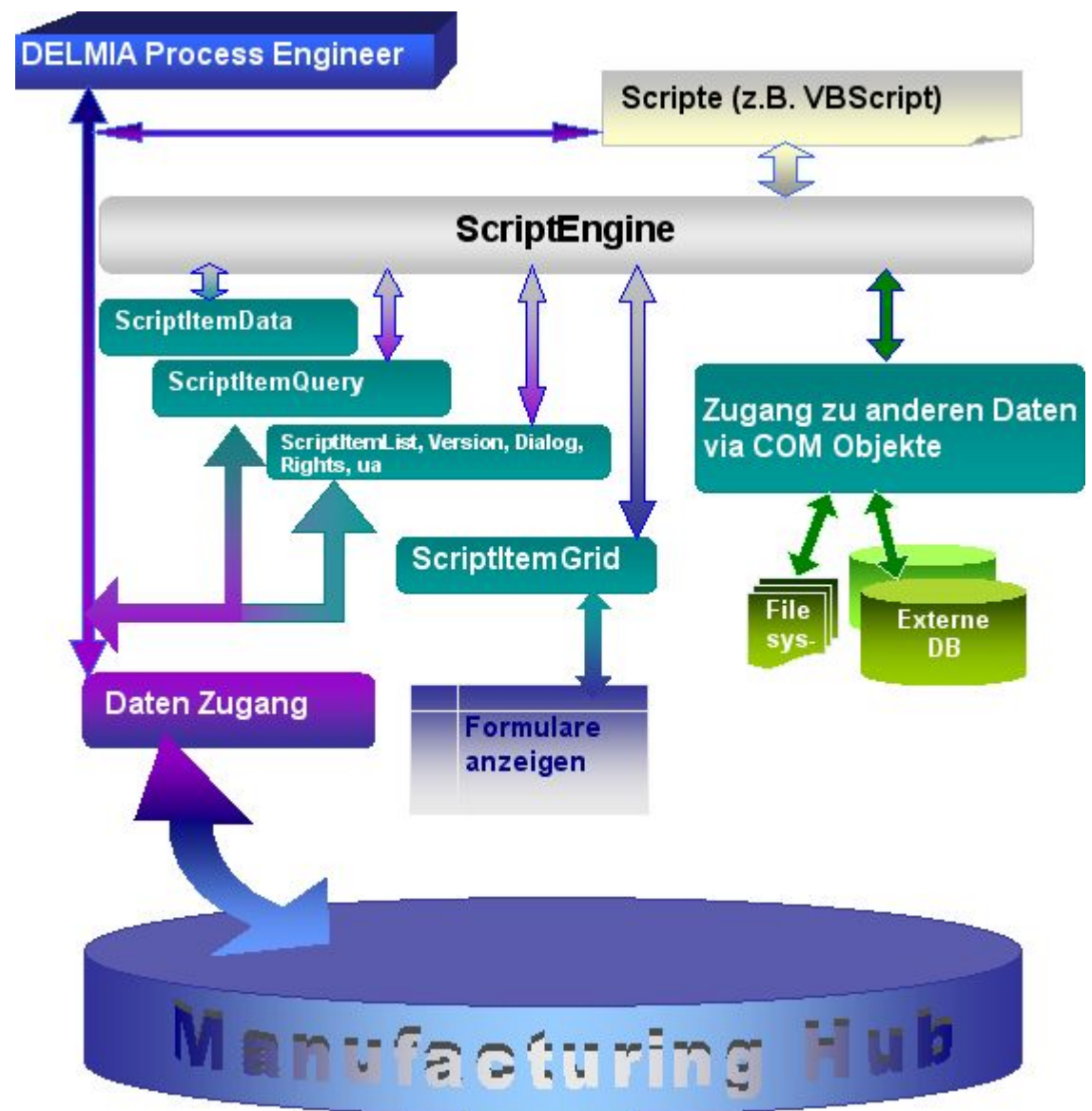
---

**Hinweis:** In den Folgeversionen des Process Engineers werden weitere named items zur Verfügung gestellt und der Funktionsumfang der Bestehenden erweitert.

Die vom DELMIA Process Engineer zur Verfügung gestellten Objekte sind nur von Skripten aus zugänglich, die innerhalb des Process Engineers ausgeführt werden. Sie können also das DELMIA Process Engineer-Objekt "data" nicht in einer beliebigen \*.vbs-Datei verwenden.

Jede dieser COM-Klassen stellt ein von IDispatch abgeleitetes duales Interface zur Verfügung, das die entsprechenden Methoden bereithält.

---



**Abbildung 2:** Zusammenwirken zwischen Skript Engine und DELMIA Process Engineer



**Hinweis:** Es ist unumgänglich, dass der Skriptschreiber neben der Kenntnis der Skriptsprache auch das Konfigurationswerkzeug des Process Engineer kennen muss, um die Attributnamen und die Namen der Eltern-Kind-Beziehungen sowie die Kinderlisten einsetzen zu können.

Was kann man mit Skripting im DELMIA Process Engineer tun?

- ➡ Beliebige Änderungen der Daten im DELMIA Process Engineer
- ➡ Beliebige mathematische Algorithmen bzw. Funktionen benutzen
- ➡ Exportieren bzw. Importieren von Daten
- ➡ Überprüfen von eingelesenen Datenstrukturen anderer Datensysteme
- ➡ Filtern von bestimmten Dateninformationen mit entsprechender Bildschirminformation

## 1.3. Skripte im DELMIA Process Engineer verwalten

### 1.3.1. Skripte, VBA-Makros und Skriptaktionen

Im DELMIA Process Engineer® können Sie Makrocodes auf unterschiedliche Arten erzeugen und verwenden. Folgende Objekte können Makrocodes verwenden

- ☐ **Skripte**
- ☐ **VBA-Makros**

Objekte des Typs **skript** werden in der Datenbank gespeichert, während **VBA-Makros** innerhalb eines Containers gespeichert werden, die zu dem Objekt *VBAProject* gehören.

- ☐ **Skriptaktionen** sind keine Skripte, sondern Objekte, die in Verbindung mit Skripten für zwei verschiedene Zwecke verwendet werden:
  - Skriptzuweisungen  
Ist die konfigurierbare Anzeige von Skripten in Kontextmenüs.  
▶ Verbindet ein Skript mit einem Konfigurations- oder Planungstyp.
  - Wirkliche (reale) Skriptaktionen  
Ist eine ereignisgesteuerte, automatische Ausführung des Skriptcodes.  
▶ Verbindet ein Skript, einen Konfigurationstyp oder einen Planungstyp mit einem Ereignis.

VBA-Makros können keiner realen Skriptaktion zugeteilt werden.

### 1.3.2. Wo befinden sich die Skripte, VBA-Makros und Skriptaktionen im PPR-Navigator?

Skripte	Projektbibliothek Systembibliothek
Skriptaktionen	Projekt Plantypensatz In jedem Plantypensatz der Systembibliothek
VBA-Makros	Sind Teil eines VBA-Projektes – nur in der Projektbibliothek

Skript-Objekte finden Sie in der **Projektbibliothek** und in der **Systembibliothek** im Ordner **Skripte**.

#### 1.3.2.1. Wie werden Skripte erstellt?

- ➔ In der Projektbibliothek finden Sie einen Ordner „Skripte“ (Projektbibliothek / Werkzeuge / Skripte). Dieser Ordner ist bei der Erstinstallation leer. Um ihn zu füllen, öffnen Sie mit einem Rechtsklick das Kontextmenü und wählen den Eintrag *Neu / Skript*.
- ⇒ Es öffnet sich der Skripteditor.
- ➔ Im Skripteditor können Sie den Namen des Skriptes festlegen und den Skriptcode eintragen.

Die gleiche Vorgehensweise benutzen Sie in der Systembibliothek.

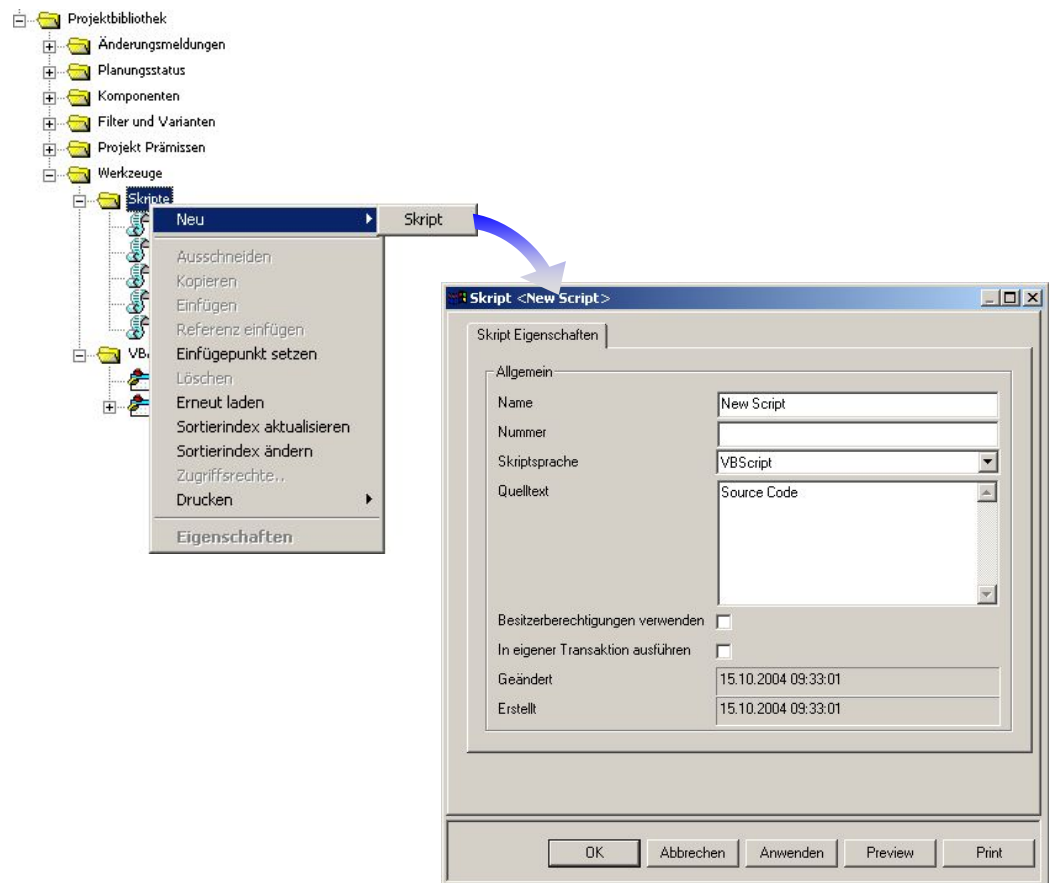


Abbildung 3: Neues Skript erzeugen

Sie sind natürlich nicht gezwungen, Ihren Skripttext unbedingt im DELMIA Process Engineer® einzugeben.

Ganz gleichgültig wo und womit Sie Ihre Skripte editieren, wichtig ist nur, dass Sie Ihren Skriptcode in das Feld "Quelltext" Ihres Skriptobjektes bringen, sei es durch direktes Editieren oder per Copy & Paste in dieses Feld.



**Hinweis:** Ist der Ordner Werkzeuge nach einer Neuinstallation nicht vorhanden, muss er mit Hilfe des Konfigurationswerkzeuges eingeblendet werden. Wie der Ordner eingeblendet wird, erfahren Sie im Handbuch Administration, Abschnitt Konfigurationsbeispiele.

## Eigenschaften festlegen

Das Dialogfeld zeigt die Eigenschaften eines Skripts. Die Registerkarte 'Skript Eigenschaften' ist aktiv. Die 'Allgemein'-Gruppe enthält folgende Felder:

- Name:** ALB Show Graphic Script
- Nummer:** (leeres Textfeld)
- Skriptsprache:** VBScript (ausgewählt)
- Quelltext:**

```
Function SA_OpenProject(id)
'Necessary for the Feature 'Show Graphic' in ALB.
'Shows the Graphic of the object, stored in the Registry.
'Then deleted the entry, so the object is shown only once.
'DELMIA GmbH 17.01.2006 - MFM
=====
Dim sOID

sOID = Data.ReadSessionData("E5Balancing", "OID_for_ShowGraphic", "HKCU")
Call Data.WriteSessionData("E5Balancing", "OID_for_ShowGraphic", "HKCU", "")

If Len(sOID) > 0 Then
    Call Graphic.ShowGraphic( sOID , "")
End If

End Function
```

Unterhalb der Quelltext-Feld befinden sich zwei Kontrollkästchen:

- ☐ Besitzerberechtigungen verwenden
- ☐ In eigener Transaktion ausführen

Die Felder 'Geändert' und 'Erstellt' zeigen das Datum 13.11.2006 und die Uhrzeit 12:47:11 bzw. 12:46:32.

Am unteren Rand befinden sich die Schaltflächen: OK, Abbrechen, Anwenden, Vorschau, Drucken, Vorheriges, Nächstes.

**Abbildung 4: Eigenschaften eines Skripts**

### 1.3.2.2. Beschreibung der Parameter

#### Name

Geben sie hier den Namen des Skripts ein. Unter diesem Namen können Sie das Skript später wiederfinden.

#### Nummer

Geben sie hier die Nummer des Skripts ein.

#### Skriptsprache

Hier legen Sie die Skriptsprache fest, in der das Skript geschrieben werden soll. Standard ist VBScript oder JScript.



### Quelltext

Hier tragen Sie den Skriptcode ein.

Die Skriptgröße (für ORACLE) ist abhängig von den Datenbankeigenschaften.

### Besitzberechtigung verwenden

Mit dieser Funktion erhalten Sie zusätzliche Rechte bei der Ausführung eines Skriptes. Lesen Sie dazu auch das Kapitel [Rechte im Skripting](#).

### In eigener Transaktion ausführen

Es ist möglich, ein Skript in einer eigenen Transaktion zu starten, d. h. das Skript arbeitet in einer vom Process Engineer unabhängigen oder weiteren Umgebung. Lesen Sie dazu auch das Kapitel [Skripte in eigener Transaktion ausführen](#).

## 1.3.3. Rechte im Skripting

Mit der Funktion *Besitzberechtigung verwenden* erhält der Benutzer zusätzliche Rechte bei der Ausführung eines Skriptes. Lesen Sie zu diesem Kapitel auch den Abschnitt Konfiguration im Administrations- Handbuch.

### 1.3.3.1. Als Skriptbesitzer ein Skript ausführen

Warum wird die Funktion *Besitzberechtigung verwenden* benötigt?

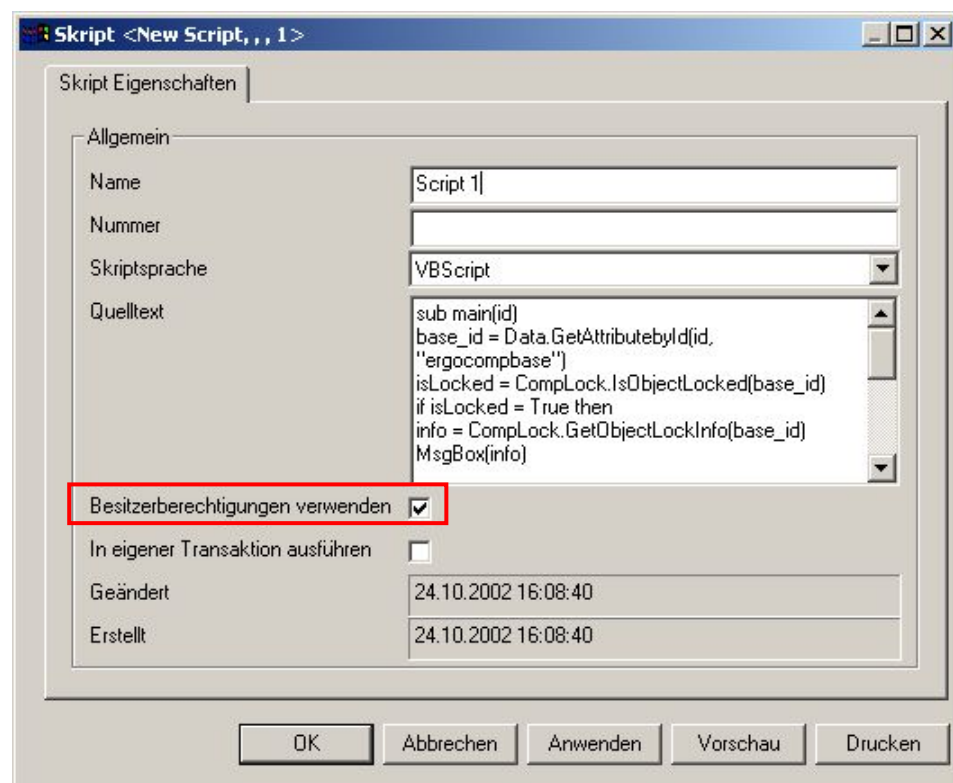


Abbildung 5: Eigenschaften eines Skripts

In den Versionen vor DPE 5.12 war der Skriptschreiber eines Skriptes auch der Eigentümer des Skriptes. In diesen Vorgänger-Versionen hatte der Eigentümer die Benutzerrechte, ein Skript zu erstellen und auszuführen, die Rechte auf bestimmte Plantypen waren dagegen eingeschränkt: Die Folge daraus war, dass es bei der Skriptausführung zu Fehlfunktionen kommen konnte.





Lesen Sie zu diesem Thema im Administrations- Handbuch das Kapitel [Benutzerverwaltung](#).

Mit der Funktion *Besitzerberechtigung verwenden* ist es nun möglich, das Skript unabhängig von der Ausführungsart zu starten, als ob der Eigentümer des Skriptes der gegenwärtige Benutzer wäre. Folgende Rechte sollten erteilt werden:

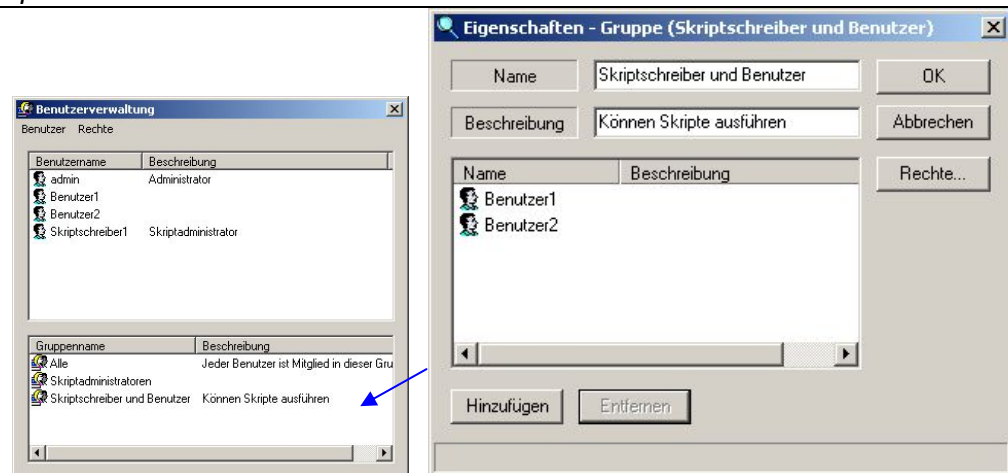
- Recht, Skripte zu erstellen, schreiben und zu ändern.
- Recht, interaktive Skripte auszuführen.

Der Skript-Schreiber hat quasi Administratorenrechte, diese Rechte werden über die Funktion *Besitzerberechtigung verwenden* auf den Ausführenden eines Skriptes übertragen.

- ➔ Aktivieren Sie die Funktion *Besitzerberechtigung verwenden* im Skriptdialog. Siehe auch: [Abbildung 5](#).



**Hinweis:** Mit der Einführung der Funktion *Besitzerberechtigung verwenden* wird der Administrator zusätzlich gefordert: Wir empfehlen daher den Administratoren, für die Benutzer eine Gruppe anzulegen, die das Recht besitzen, interaktive Skripte zu schreiben und auszuführen.



**Abbildung 6: Beispiel aus der Benutzerverwaltung**

*Gibt es kein Problem, wenn einem normalen Benutzer gewährt wird, eigene Skripte zu erstellen?*

Nicht wirklich. Jedes Mal, wenn ein normaler Benutzer ein Skript erstellt, ist er/sie die EigentümerIn des Skriptes und selbst wenn *Besitzerberechtigung verwenden* („RunAsOwner“) aktiviert ist, werden nur die Rechte dieses Benutzers beachtet.

### 1.3.4. Skripte in eigener Transaktion ausführen

In eigener Transaktion ausführen ☐

Es ist jetzt möglich, ein Skript in einer eigenen Transaktion zu starten. Dies ist besonders nützlich, wenn Sie ein Skript ausführen, das kritische Änderungen herbeiführen kann. Wenn bei der Ausführung Fehler auftreten, werden die geänderten Daten nicht gespeichert. Diese Funktion kann unabhängig für jedes Skript im Dialog *Eigenschaften* eines Skripts aktiviert werden.

#### Was bedeutet es, ein Skript in einer eigenen Transaktion aufzurufen?

- Zur Laufzeit des Skriptes werden die Änderungen, die das Skript vornimmt, nicht gespeichert.
- Treten Fehler während der Skriptausführung auf, wird das Skript ohne Abspeichern beendet. Änderungen, die bis zum Auftreten des Fehlers anfielen, werden nicht in der Datenbank gespeichert.
- Erst nachdem das Skript fehlerfrei beendet wurde, werden die Änderungen gespeichert.

Ist die Funktion aktiviert und das Skript aktiv, werden Sie solange keine sichtbaren "Online-" Änderungen im PPR Navigator wahrnehmen, bis Sie nicht einen Befehl `Data.CommitTransaction` in Ihrem Skript selbst aufrufen.

## 1.4. Skripte ausführen

Sie haben unterschiedliche Möglichkeiten, Skripte zu starten.

### 1.4.1. Starten eines Skriptes über „Skript ausführen“

Ein Skript wird über das Kontextmenü eines Projektes oder einer Projektkomponente ausgeführt. Sie finden dort jeweils einen Menüpunkt „**Skript ausführen**“. Wenn Sie diesen Menüpunkt auswählen, erhalten Sie eine Auswahl aller dem Projekt zugeordneten Skripte, auf welche Sie das Recht zur Ausführung besitzen.

Haben Sie in den *Einstellungen* den Eintrag **Schnelle Skriptausführung aktiviert** ☒ aktiviert, wird in der Bibliothek das selektiert Skript direkt ausgeführt.

Diese Option sollte nur von erfahrenen Anwendern benutzt werden, da es bei unsachgemäßer Verwendung dieser Option, zu fehlerhaften Daten oder zu Datenverlust kommen kann.

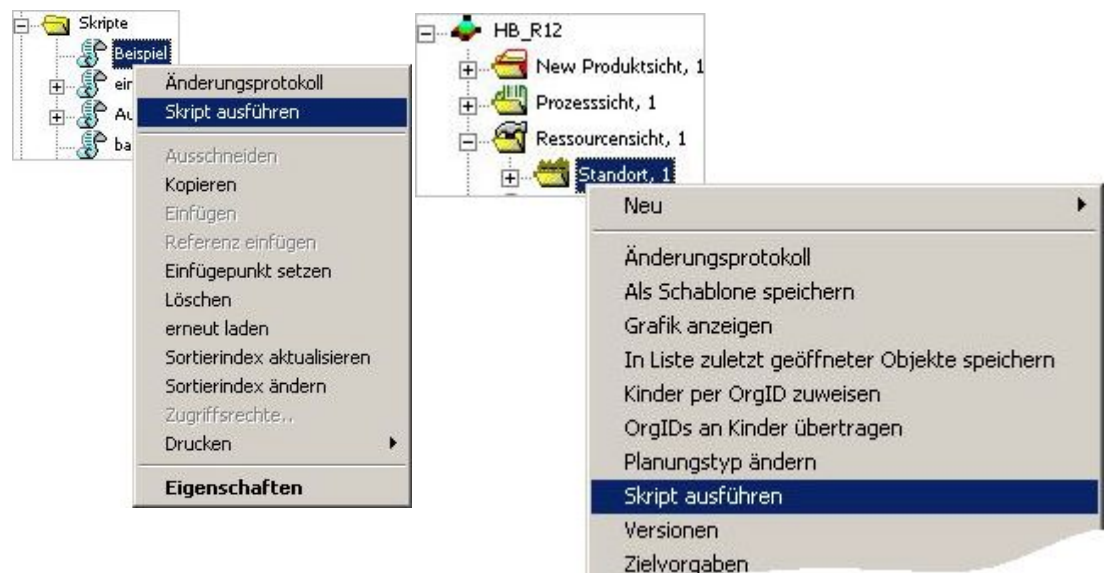


Abbildung 7: Möglichkeit, ein Skript zu starten

- ⇒ Es öffnet sich ein Fenster, in dem alle verfügbaren Skripte in einer Liste angezeigt werden. Wählen Sie aus diesem Menü das von Ihnen gewünschte Skript aus. Wenn Sie die Auswahl mit OK bestätigen, wird das Skript gestartet.

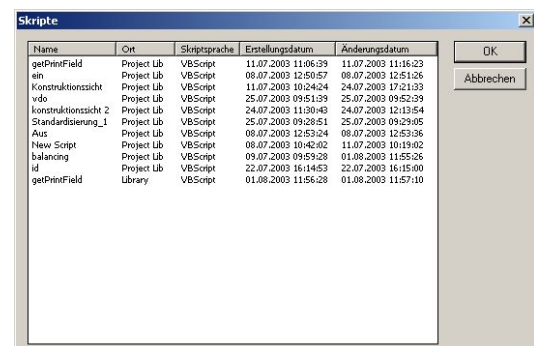


Abbildung 8: Liste der zur Verfügung stehenden Skripte



**Hinweis:** Es ist absolut nicht sinnvoll, ein beliebiges Skript von einer beliebigen Komponente aus zu starten. Deshalb ist die Option „Skript ausführen“ standardmäßig deaktiviert.

## 1.4.2. Starten eines Skriptes im interaktiven Modus

Bisher wurden die Skripte über den Eintrag „Skript ausführen“ im Kontextmenü einer Komponente gestartet. Diese Art, ein Skript zu starten, sollte nicht als Standard dienen. Wenn Sie z. B. viele Skripte haben, ist das Suchen nach dem richtigen Skript zeitaufwendig und fehleranfällig. Deshalb sollten Sie die nachfolgend dargestellten Methoden, ein Skript zu starten der vorhin beschriebenen Methode vorziehen.

### 1.4.2.1. Skripte automatisch starten mit Hilfe von Skriptaktionen

Neben dem oben erwähnten Aufruf „Skript ausführen“ im Kontextmenü einer Komponente haben Sie weitere Möglichkeiten, Skripte aus dem Kontextmenü zu starten.

Den Ordner Skriptaktionen haben Sie schon im Abschnitt [Skripte, VBA-Makros und Skriptaktionen](#) kennen gelernt. Denselben Ordner verwenden Sie auch dazu, um Skripte automatisch zu starten. Beim automatischen Starten von Skriptaktionen werden die Skripte zusammen mit bestimmten Ereignissen gestartet. Unter Ereignissen versteht man beispielsweise eine neue Komponente erzeugen, kopieren oder auch zu verschieben (siehe auch: [Abbildung 12](#)). Beim Ausführen von Skriptaktionen wird entweder der eingegebene Typ oder der Plantyp berücksichtigt. Diese Eingaben machen Sie, wenn Sie eine Skriptaktion erzeugen (siehe auch: [Abbildung 10](#)).

#### So gehen Sie vor:

- ➔ Selektieren Sie im Plantypensatz des Projekts den Ordner „Skriptaktionen“.
- ➔ Erstellen Sie eine neue Skriptaktion über den Eintrag „Neu“ im Kontextmenü.

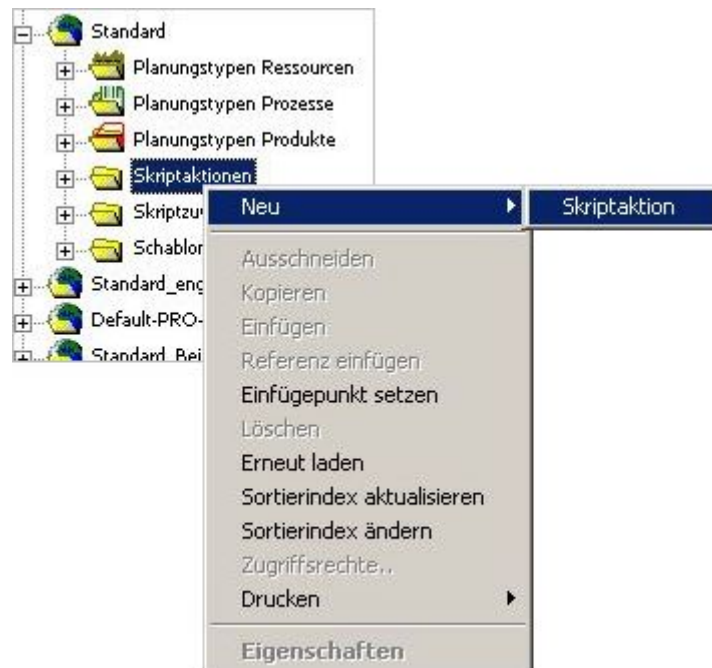


Abbildung 9: Neue Skriptaktion

- ➔ Den Dialog öffnen Sie mit einem Doppelklick auf die neu angelegte Skriptaktion. Tragen Sie im geöffneten Dialog einen Namen ein

- ➔ Wählen Sie danach unter *Script/VBA-Macro* das Skript aus, für das die Skriptaktion ausgeführt werden soll (siehe auch: [Abbildung 10](#)).
  - ➔ Geben Sie unter *Type/Plantype* bei Konfigurationstyp, z. B. ergoproject ein. Speichern und verlassen Sie die Skriptaktion mit *OK*.
- ⇒ Dieses Skript wird nun bei dem gewählten Aktionstyp immer gestartet.

The image shows a Windows-style dialog box titled "Eigenschaften Skriptaktion". It contains several input fields and controls:

- Name:** A text box containing "New Script Action".
- Skript:** A dropdown menu that is currently empty.
- Konfigurationstyp/Plantyp:** A section containing two dropdown menus, both of which are empty.
- Eigenschaften Aktion:** A section containing:
  - Aktionstyp:** A dropdown menu that is empty.
  - Reihenfolge der Ausführung:** A text box containing the number "1".
  - Vor/Nach:** Two radio buttons. The "Vor" button is unselected, and the "Nach" button is selected.
- Aktiv:** A checkbox that is checked.
- Buttons:** "OK" and "Abbrechen" buttons are located at the bottom right of the dialog.

**Abbildung 10: Eigenschaften einer Skriptaktion**

## Beschreibung der Parameter

Die Beschreibung der einzelnen Parameter für die Eingabe finden sie in der nachfolgenden Tabelle. Siehe auch: [Abbildung 11](#).

Attribut	Wert	Bemerkung
<b>Name</b>	Für Skriptaktion	Beschreibung der Skriptaktion
<b>Skript</b>	Skript zum Ausführen	Hier wählen sie den Namen des Skriptes, dem eine Aktion zugewiesen werden soll.
<b>Plantyp</b>	Plantyp, für den das Skript ausgeführt werden soll	Spezifizieren Sie hier entweder den Plantyp oder den Typ.
<b>Typ</b>	Typ, für den das Skript ausgeführt werden soll	Hier geben Sie den <b>Typ</b> an, wenn eine Auswahl von Plantypen vom selben Typ abzudecken ist. Einem <b>Typ</b> können durchaus mehrere Plantypen zugeordnet sein oder wenn eine Skriptaktion auf einem Objekt ausgeführt werden soll, das keinen Plantyp hat.
<b>Actionstyp</b>	Angabe des Typs einer Skriptaktion	Dieser Typ muss zum eingebundenen Skript verträglich sein. Zum Beispiel, "Copy" muss einer Funktion "sa_copy" im Skript entsprechen.
<b>Vor/Nach</b>	Angabe, ob das Skript vor oder nach dem Ereignis ausgeführt werden soll.	<b>Achtung:</b> Im Falle von „Vor“ kann das Skript die laufende Aktion durch die Rückgabe von "False" blockieren.  <b>Hinweis:</b> Einige Ereignisse können nur eine "Vor" oder "Nach" Skriptaktion ansteuern, z. B. sa_delete".
<b>Aktiv</b>	Deaktivieren oder aktivieren der automatischen Ausführung eines Skriptes	Nur ab PE 5.9 .
<b>Reihenfolge der Ausführung</b>	Eine ganze Zahl. Standard ist 1.	Wenn mehr als eine Skriptaktion für dieselbe Aktion und denselben Plantyp (Typ) angegeben ist, so kann die Reihenfolge der auszuführenden Skripte hier festgelegt werden, falls die Skripte nicht in willkürlicher Reihenfolge ausgeführt werden sollen.

**Abbildung 11: Tabelle mit Parametern für die Eingabe bei Skriptaktionen**

- Um eine Skriptaktion zu starten, benötigen Sie die dazugehörige Funktion.
- Nachfolgend werden die Ereignisse, die Skriptaktionen auslösen, aufgeführt: Siehe auch [Abbildung 12](#).

Ereignis	Aktionen in Abhängigkeit ihres Aufrufs
Eine neue Komponente erstellen ( <b>Neu ...</b> )	NewChild/before AddChild/after SetAttribute/before SetAttribute/after New/After
Komponente(n) <b>kopieren</b>	Copy/before SetAttribute/before SetAttribute/after Copy/After
Komponente(n) <b>verlinken</b>	Link/before Link/after
Komponente(n) <b>verschieben</b>	Move/before Move/after
<b>Referenz löschen</b> (subcompitem/relationship)	RemoveChild/before RemoveChild/after
<b>Komponenten löschen</b> (ergocompbase/ergoitem / Andere)	Delete/Before
Eigenschaften-Dialog öffnen	InitProperties/After
Beim Ändern des Wertes einer Combobox oder einer Listbox innerhalb eines Eigenschaften-Dialogs	ChangeProperties/After
Eigenschaften <b>ändern</b>	SetAttribute/before SetAttribute/after
Eine <b>neue Version</b> erstellen oder eine neue Version erproben	Neue Version/before NewVersion/after
Den <b>Planungsstatus ändern</b> oder eine Version prüfen	SetPlanningState/before SetPlanningState/after
Beim <b>Öffnen eines Projektes</b>	OpenProject/after
Vor dem <b>Schließen eines Projektes</b>	CloseProject/before
<b>Drucken</b>	Print/before Print/after
Austaktung ( <b>Balancing</b> ) <b>starten</b>	StartBalancing/After
Austaktung ( <b>Balancing</b> ) <b>speichern</b>	SaveBalancing/After
<b>Prozess</b> in die Austaktung <b>verschieben</b>	DropProcessToBalancing/After
Öffnen oder Bearbeiten eines Grafik Objectes.	Open Graphic/Before

**Abbildung 12: Tabelle für Ereignisse**

#### 1.4.2.2. Skriptaktionen allgemein

Skriptaktionen werden bei bestimmten Ereignissen automatisch ausgeführt, im Gegensatz zu Skriptzuweisungen, die manuell angestoßen werden. Trifft ein bestimmtes Ereignis ein, wird ein Skript, das über eine Skriptaktion diesem Ereignis zugeordnet wurde, automatisch ausgeführt (z. B. ein neues Objekt einer Struktur hinzufügen oder mit einem anderen Objekt verknüpfen).

Reale Skriptaktionen werden jedes Mal beim Starten eines bestimmten Ereignisses ausgeführt. Dabei kann das Skript **vor** bzw. **nach** diesen Ereignissen gestartet werden.

Reale Skriptaktionen unterscheiden sich in einem weiteren Punkt von Skriptzuweisungen. Während Sie mit Skriptzuweisungen ‚normale‘ Skripte oder VBA-Makros steuern, müssen reale Skriptaktionen eine Funktion besitzen, die dem Ereignis entsprechen, an welches die Skriptaktion geknüpft werden soll.

- Sie fügen beispielsweise ein neues Objekt einer Struktur hinzu. Beim Anlegen der Skriptaktion haben Sie dann zuvor festgelegt, ob das Skript **vor** diesem Vorgang oder **danach** ausgeführt werden soll. Das Skript selbst muss die Funktion `function sa_new(new_object_id)` enthalten.

Skriptaktionen werden genauso erzeugt wie Skriptzuweisungen. Zusätzlich müssen Sie beim Anlegen im Eigenschaftsdialog den *Aktionstyp* auswählen. Alle weiteren Eingaben im Eigenschaftsdialog entsprechen der Eingabe bei der Skriptzuweisungen.

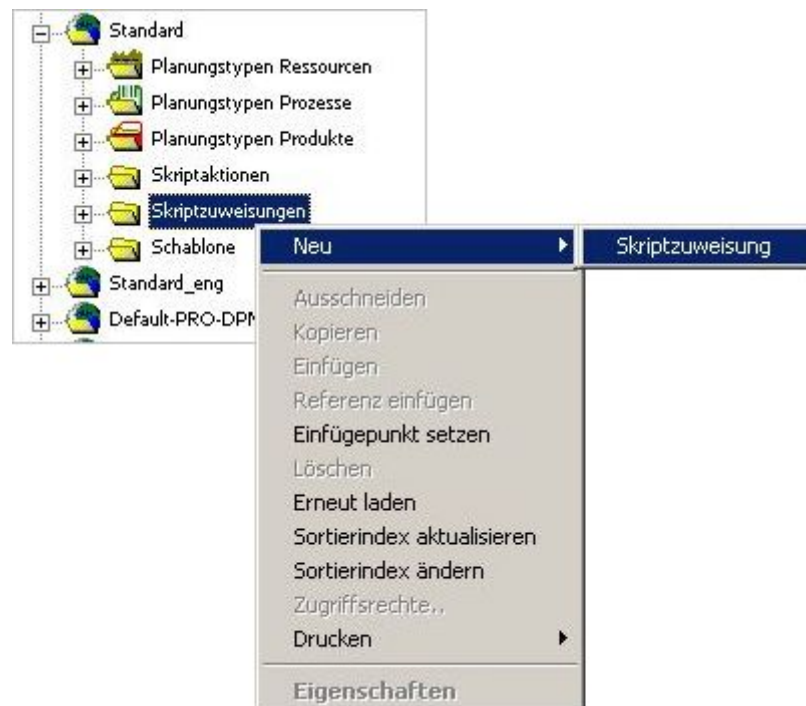
Für die Beschreibung der einzelnen Ereignisse, die Skriptaktionen auslösen, siehe auch: [Liste aller Skriptaktionen](#).



### 1.4.3. Skriptzuweisungen anlegen

Mit Skriptzuweisungen wird eine Verbindung zwischen einem ausgewählten Plantypen oder Typ zu einem Skript oder VBA-Makro hergestellt, das über das Kontextmenü des Plantyps bzw. Typs unter dem Eintrag *Skripte oder VBA-Makros* ausgeführt wird. Diesen Eintrag im Kontextmenü erhalten Sie aufgrund der hergestellten Verbindung zu einem Plantyp bzw. Typ im Eigenschaftsdialog der Skriptaktion.

- ➔ Selektieren Sie im Plantypensatz des Projekts oder in der Systembibliothek den Ordner „Skriptzuweisungen“.
- ➔ Erstellen Sie eine neue Skriptzuweisungen über den Eintrag „Neu“ im Kontextmenü.



**Abbildung 13: Neue Skriptzuweisung**

- ➔ Geben Sie unter *Type/Plantype* bei Konfigurationstyp, z. B. ergoproject ein. Speichern und verlassen Sie die Skriptaktion mit **OK**.
- ⇒ Sie können dieses Skript nun über das Kontextmenü des Projektknotens starten.

Falls Sie anstelle eines Konfigurationstyp einen Planungstyp (z. B. Baugruppe) eingegeben haben, so wird das Script *nicht* über das Kontextmenü des Projektknotens, sondern in Kontextmenü einer Baugruppe gestartet. Siehe auch: [Abbildung 14](#).



**Abbildung 14: Script über das Kontextmenü auf den Produktknoten starten**

Weitere Erklärungen zu den Eingabeparameter finden Sie im Kapitel, siehe auch: [Beschreibung Eingabeparameter für Skriptzuweisungen](#).



**Achtung:** Skriptzuweisungen sollten ausschließlich von einem Administrator oder einem gleichberechtigten Mitarbeiter ausgeführt werden.

**Abbildung 15: Skriptzuweisungen anlegen – Eigenschaftsdialog**

## Beschreibung Eingabeparameter für Skriptzuweisungen

Attribut	Wert	Bemerkung
<b>Name</b>	Für Skriptzuweisung	Beschreibung der Skriptzuweisung
<b>Bezeichnung</b>	Für Kontextmenü	Diese Zeichenfolge wird als Eintrag im Kontextmenü angezeigt.
<b>Skript</b>	Skript zum Ausführen	Hier wählen sie den Namen des Skriptes aus, dem eine Aktion zugewiesen werden soll.
<b>VBA Macro</b>	VBA-Macro zum Ausführen	Hier wählen Sie den Namen des VBA-Makros aus, dem eine Aktion zugewiesen werden soll.
<b>Planungstyp</b>	Planungstyp, für den das Skript ausgeführt werden soll	Spezifizieren Sie hier entweder den Planungstyp oder den Typ.
<b>Konfigurationstyp</b>	Typ, für den das Skript ausgeführt werden soll	Hier geben Sie den <b>Typ</b> an, wenn eine Auswahl von Planungstypen vom selben Typ abzudecken ist. Einem <b>Typ</b> können durchaus mehrere Planungstypen zugeordnet sein.  Oder wenn eine Skriptaktion auf einem Objekt ausgeführt werden soll, das keinen Planungstyp hat.
<b>sichtbar</b>	Deaktiviert oder aktiviert den Eintrag eines Skriptes oder VBAMakros, in einem Dialog oder Kontextmenü.  <b>Im Kontextmenü anzeigen</b> (Display in context menu)  <b>Im Dialog anzeigen</b> (Display in properties dialog)	<p>► Ist kein Kontrollkästchen aktiviert, kann <b>kein</b> Skript über das Kontextmenü oder im Eigenschaftsdialog aufgerufen werden, obwohl eine Skriptzuweisung angelegt ist. Die gleiche Wirkung erzielt man durch das Deaktivieren des Kontrollkästchens <b>sichtbar</b>, auch wenn die anderen beiden Kontrollkästchen aktiviert sind.</p> <p>► Mit der Aktivierung des Kontrollkästchens <b>Im Kontextmenü anzeigen</b> wird das Skript über das Kontextmenü eines Objektes aufgerufen.</p> <p>► Mit der Aktivierung des Kontrollkästchens <b>Im Dialog anzeigen</b> wird das Skript über einen Button im Eigenschaftsdialog eines Objektes aufgerufen. Ab der Version PE 5.16SP6 wird bei der Anzeige im Eigenschaftsdialog zwischen Seiten, Gruppen und Attributen unterschieden. Die Darstellungsart nehmen Sie unter <i>Options</i> vor.</p> <p>Skriptzuweisungen können nur für Planungstypen definiert werden.</p> <p>Um ein Skript auf einer Seite, Gruppe oder einem Attribut auszuführen, muss diese Seite, Gruppe oder Attribut dafür konfiguriert sein. Im Konfigurationsmanager finden Sie die Eigenschaft <b>Makro kann zugewiesen werden</b>. Diese Eigenschaft muss auf <b>Ja</b> stehen. Um den Button zu platzieren, wählen Sie in der Eigenschaft <b>Position</b> die Buttonposition aus.</p>

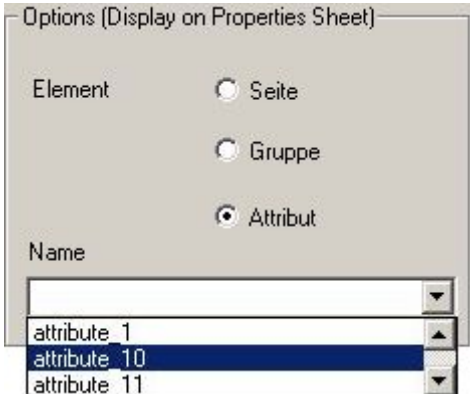
<b>Options</b> <b>(Display on Properties Sheet)</b>	<b>Seite</b> <b>Gruppe</b> <b>Attribut</b>	<p>Hier legen Sie fest wo die Skriptaktion ausgeführt werden soll. Je nach Auswahl, werden in dem darunterliegenden Auswahlfeld andere Werte auflisten. Im Beispiel wird die Skriptaktion am Attribut 10 ausgeführt.</p>  <p>Wenn Sie die Skriptaktion auf einer Seite oder Gruppe ausführen wollen, werden Ihnen in dem Auswahlfeld die konfigurierten Seiten (auch abgeleitete) des ausgewählten Planungstypen zur Auswahl angeboten.</p>
<b>Skript Ausführung</b>	<b>For each object</b> <b>Once for all objects</b>	<p>Diese Einstellung ist nur bei einer Mehrfachselektion wichtig.</p> <p>Wurden mehrere Objekte in einer Liste selektiert, legen Sie mit Hilfe dieses Eintrags fest ob die <b>main-Methode</b> des Skriptes für alle selektierten Objekte ausgeführt werden soll, oder ob für jedes einzelne Objekt in der Selektion die <b>valuation-Methode</b> des Skriptes aufgerufen werden soll. Die Methode <b>valuation</b> muss dann aber auch im Skript vorhanden sein, sonst erhält man eine Fehlermeldung. Siehe auch <a href="#">valuation</a>.</p>

Abbildung 16: Tabelle für die Eingabeparameter einer Skriptzuweisungen

## Verschiedene Plantypen

Wenn mehrere Objekte von **unterschiedlichen** Plantypen im PPR Navigator Ansichtsliste ausgewählt werden, so müssen die Skriptanweisungen denselben Namen und dieselbe Beschreibung haben, und ebenso dasselbe Skript ausführen. Wenn eine dieser Regeln nicht erfüllt wird, dann wird die Skriptanweisung im PPR Navigator nicht angezeigt.

### 1.4.3.1. Die Methode **valuation**

Die Methode **valuation** wird in Verbindung mit einer Skriptzuweisung und der dabei verwendeten Einstellung **Skript ausführen** verwendet.

Bei der Einstellung **Skript ausführen** haben Sie zwei Auswahlmöglichkeiten:

- **For each object:** Die **main-Methode** wird als Startmethode für jedes selektierte Objekt aufgerufen.

- **Once for all objects:** Die **valuation-Methode** wird für jedes selektierte Objekt aufgerufen. Die **valuation-Methode** ist dabei die Startmethode, die **main-Methode** wird als Startmethode ignoriert oder wird als normale Subroutine betrachtet.
- ⇒ Soll ein Skript bei beiden Einstellungen funktionieren, muss es eine **main-Methode** und eine **valuation-Methode** besitzen.

### Beispiel

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

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

Fehlt die **valuation** Methode und die Einstellung einer Skriptzuweisung ist auf **Once for all objects** gestellt, wird das Skript mit der nachfolgenden Fehlermeldung abgebrochen.

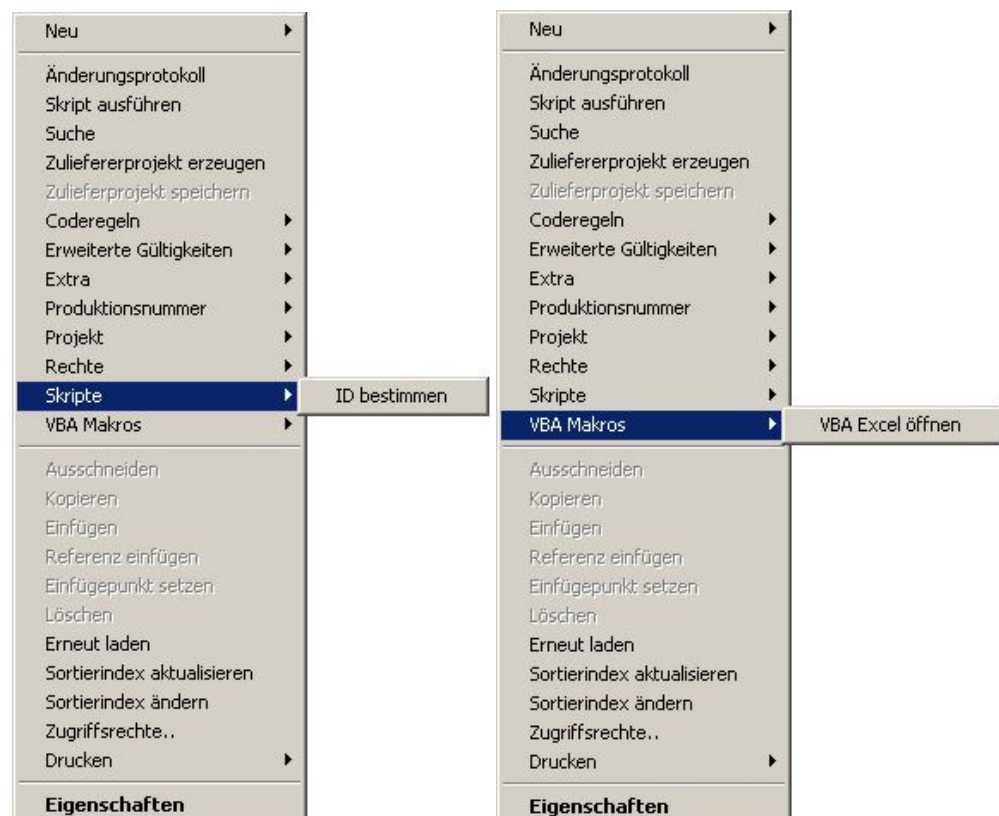


**Abbildung 17: Fehlermeldung beim Fehlen der Methode valuation und der Einstellung Once for all objects**

### 1.4.3.2. Skript über das Kontextmenü ausführen

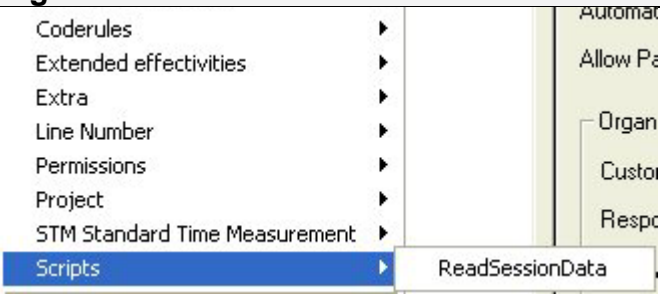
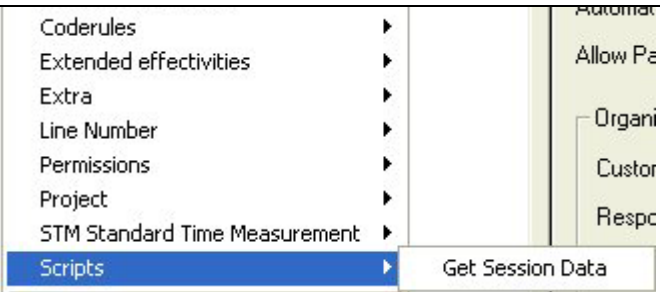
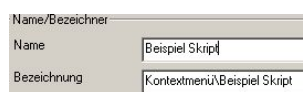

Das Skript oder VBA-Makro führen Sie über das Kontextmenü des zugeordneten Plantyps bzw. Typs aus.

- ➔ Selektieren Sie den *Plantype* bzw. *Typ* im PPR-Navigator und öffnen danach das Kontextmenü.
- ➔ Im Kontextmenü finden Sie entweder den Eintrag Skripte (bei zugeordneten Skripten) oder VBA-Makros (bei zugeordneten Makros).
- ➔ Um ein Skript bzw. ein Makro auszuführen, klicken Sie auf den jeweiligen Eintrag (z. B. auf den Namen des Skripts, *ReadChildren*).



**Abbildung 18: Skript oder VBA-Makro über Kontextmenü ausführen**

Sie haben die Möglichkeit Skripte nicht mehr nur über das Kontextmenü ‚Skripte‘ zu starten, sondern auch Funktionsbezogen, in einem eigenen Kontextmenü. Dafür müssen Sie nur in der Bezeichnung zwei Zeichenfolgen durch ein Backslash trennen. Der Backslash dient also als Trennzeichen. In der nachfolgenden Tabelle werden die Unterschiedlichen Aufrufmöglichkeiten nochmals zusammengefasst.

Bezeichnung	Beispiel	Ergebnis
Leer	(Der Name eines Skripts oder eines VBA-Makros. Im Beispiel "ReadSessionData".)	
Eine Zeichenfolge	Get Session Data	
Eine Zeichenfolge	Kontextmenü\Beispiel Skript 	



### 1.4.3.3. Skript im Eigenschaftsdialog eines Objektes ausführen

- ➔ Um ein Skript oder ein VBA-Makro im Eigenschaftsdialog eines Objektes aufzurufen, muss beim Erstellen der Skriptzuweisung die Eigenschaft **sichtbar**/ *Im Dialog anzeigen* / *Seite* aktiviert werden.
- ➔ Danach muss die Seite ausgewählt werden, auf welcher das Skript aufgerufen werden soll.

#### Beispiel

The screenshot shows the 'Eigenschaften Skriptzuweisung' dialog box. It is divided into several sections:
 

- Name/Bezeichner:** Both 'Name' and 'Bezeichnung' fields contain 'Page ScriptCommand'.
- Skript/VBA Makro:** The 'Skript' dropdown is set to 'ID', and the 'VBA Makro' field is empty.
- Konfigurationstyp/Plantyp:** The 'Planungstyp' dropdown is set to 'Part', and the 'Konfigurationstyp' field is empty.
- Zuweisungseigenschaften:** Three checkboxes are checked: 'sichtbar', 'Im Kontextmenü anzeigen', and 'Im Dialog anzeigen' (circled in red). The 'Skript Ausführung' dropdown is set to 'For each object'.
- Options (Display on Properties Sheet):** Under the 'Element' radio button, 'Seite' is selected. Below this, a list box shows '3D-View', 'Effectivity', and 'General' (circled in red). At the bottom right, the 'OK' button is circled in red.

- ➔ Selektieren Sie den *Planungstyp* im PPR-Navigator und öffnen danach den Eigenschaftsdialog.
- ⇒ Im Eigenschaftsdialog, unter dem Reiter Allgemein, finden Sie einen Button mit der Beschriftung *Page ScriptCommand*. Siehe auch [Abbildung 19](#).
- ➔ Um ein Skript bzw. ein Makro auszuführen, klicken Sie auf den jeweiligen Button (z. B. auf den Namen des Skripts, *Page ScriptCommand*). Es könne mehrere Skripte in einem Eigenschaftsdialog anzeigen werden.



Abbildung 19: Skriptaufzuruf im Eigenschaftsdialog eines Objektes

### Beispiel einer Skriptzuweisung für ein Attribut

Im Beispiel wird eine Skriptzuweisung für den Planungstyp Teil (Part) erstellt.

- ➊ Zuerst wird überprüft ob das Attribut für die Zuweisung von Skripten konfiguriert ist:

- ➡ Im nächsten Schritt wird die Skriptzuweisung erzeugt

The dialog box 'Eigenschaften Skriptzuweisung' contains the following fields and options:

- Name/Bezeichner:**
  - Name: Example script command
  - Bezeichnung: Example script command
- Skript/VBA Makro:**
  - Skript: ID
  - VBA Makro: (empty)
- Konfigurationstyp/Plantyp:**
  - Planungstyp: Part
  - Konfigurationstyp: (empty)
- Zuweisungseigenschaften:**
  - ☒ sichtbar
  - ☐ Im Kontextmenü anzeigen
  - ☒ Im Dialog anzeigen
  - Skript Ausführung: For each object
- Options (Display on Properties Sheet):**
  - Element: ☐ Seite, ☐ Gruppe, ☒ Attribut
  - Name: attribute\_10

Buttons: OK, Abbrechen

- ➡ Im letzten Schritt wird ein Objekt vom Planungstyp Teil geöffnet und überprüft ob die Skriptaktion richtig erzeugt wurde:

The dialog box 'TPZ-Kurve' contains the following fields and options:

- TPZ-Kurve: (empty)
- Planungsvariante: (empty)
- Änderungsprotokoll erstellen: ☐
- Use for macro Attribute 10:** (highlighted with a red box, followed by an ellipsis button)
- Kosten: (empty)
- Materialeinzelkosten: 0,00 €

## 1.4.4. Skripte automatisch starten

### 1.4.4.1. Mit Skriptaktionen Skripte automatisch starten

Lesen Sie dazu das Kapitel

[Skripte automatisch starten mit Hilfe von Skriptaktionen](#) und [Skripte im Batch-Modus starten](#).

## 1.4.5. Skripte im Batch-Modus starten

Falls Skripte sehr zeitintensiv sind oder regelmäßig ausgeführt werden sollen, haben Sie jetzt die Möglichkeit, Skripte im Batch-Modus (Stapelverarbeitungs-Modus) zu starten, d. h. Sie können mit Skripten direkt auf den DELMIA Process Engineer zugreifen.

Ab der Version PE 5.14 haben Sie eine zusätzliche Möglichkeit Skripte im Batch-Modus aufzurufen. Dabei müssen Sie den Namen eines Skriptes angeben welches den ausführbaren Skriptcode enthält.

### 1.4.5.1. Batch-Aufruf mit Hilfe einer Datei

#### Bis Version PE5.15

Dazu benötigen Sie die Skriptdatei. Damit starten Sie das Skript dann folgendermaßen:

➔ Ausführbare Datei starten: `\\PPRClient\program\bin\DPFFrame.exe`

➔ Parameterangabe

`/username:[login] /userpwd:[pwd]`

`/startobject:ScriptEngineLoader.ScriptEngineLoaderManager "[filepath] [entry function] [script language]"`

#### **Beispiel:**

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

- Der zweite und dritte Parameter sind optional. Standardmäßig wird von der *main* Eingangsfunktion und der *vbscript* Sprache ausgegangen. Deshalb müssen diese Parameter nicht explizit angegeben werden.
- Die Eingangsfunktion muss keine Eingangsparameter besitzen! Dadurch unterscheidet sich der Batch-Aufruf von den Standard Process Engineer Skripten, in welchen die *main* Eingangsfunktion als Parameter die ID benötigt.

#### Ab der Version PE5.15

```
"<USERNAME>login</USERNAME> <E5_PWD>pwd</E5_PWD>
<STARTOBJECT>ScriptEngineLoader.ScriptEngineLoaderManager</START
OBJECT> [filepath] [entry function] [script language]"
```

#### **Beispiel:**

```
\\PPRClient\program\bin\DPFFrame.exe
"<USERNAME>admin</USERNAME> <E5_PWD>admin</E5_PWD>
```

```
<STARTOBJECT>ScriptEngineLoader.ScriptEngineLoaderManager</START
OBJECT> c:\temp\myScript.txt|main|vbscript"
```

**Hinweis:**

- ▶ Die Parameter *[entry function]* und *[script language]* sind optional. Die Standard-Einsprungmethode ist **main**, die Standard-Skriptsprache ist **vbscript**.
- ▶ Die Startmethode muss **keine** Eingangsparameter besitzen! Dies unterscheidet sie von ‚normalen‘ Process Engineer Skripten, deren Startmethode immer ein Parameter besitzen muss (siehe auch 1.5).
- ▶ Beachten Sie bitte, dass in der Version PE5.14 die Anführungszeichen den Block "[filepath][entry function][script language]" umschließen, während in der PE5.15 Version, die Anführungszeichen die ganze Befehlszeile nach dem Programmnamen umschließen..
- ▶ In der Version PE5.14 ist das Leerzeichen zwischen dem Anfangsobjektnamen und dem ersten Anführungszeichen obligatorisch, in PE5.15 ist das Leerzeichen zwischen dem </STARTOBJECT > und dem Dateinamen obligatorisch.

**1.4.5.2. Batch-Aufruf mit Hilfe eines Objektes****Bis Version PE5.15**

Dazu benötigen Sie die Skript Objekt ID. Damit starten Sie das Skript dann folgendermaßen:

- ➡ Ausführbare Datei starten: \PPRClient\program\bin\DPFFrame.exe
- ➡ Parameterangabe

```
/username:[login] /userpwd:[pwd]
/startobject:ScriptEngineLoader.ScriptEngineLoaderManager "[script object
ID]||[main entry object ID]"
```

**Beispiel:**

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

**Ab der Version PE5.15**

```
"<USERNAME>login</USERNAME> <E5_PWD>pwd</E5_PWD>
<STARTOBJECT>ScriptEngineLoader.ScriptEngineLoaderManager</START
OBJECT> [script object ID]||[main entry object ID]"
```

**Beispiel:**

```
\\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)"
```



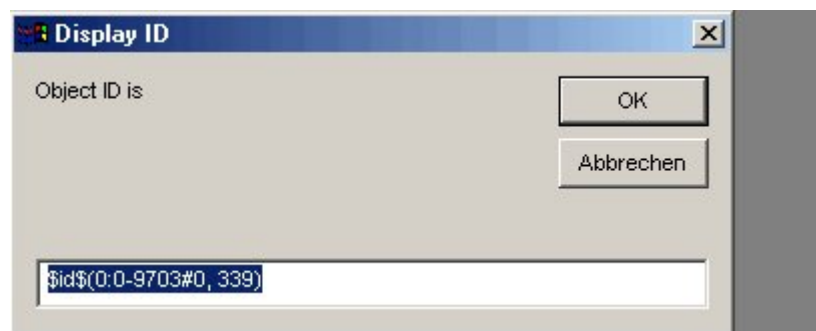
**Hinweis:**

► Der Parameter *[main entry object ID]* ist optional. Wenn eine Objekt-ID erforderlich ist, dann verketteten Sie diese zur Skript-Objekt-ID mit Hilfe eines "|" (pipe) Zeichens. Ist dies nicht der Fall, tragen Sie nur die Skript-Objekt-ID ein.

► Beachten Sie bitte, dass in der Version PE5.14 die Anführungszeichen den Block "[filepath][entry function][script language]" umschließen, während in der PE5.15 Version, die Anführungszeichen die ganze Befehlszeile nach dem Programmnamen umschließen..

Die Objekt-ID erhalten Sie am schnellsten mit Hilfe des nachfolgenden Skripts. Vorteil dieses Skriptes ist die Kopiermöglichkeit der gefundenen ID in der Ausgabebox. Sie können dieses Skript auf jedem Knoten aufrufen.

```
sub main(i d)
    val = InputBox("Object ID is", "Display ID", i d)
end sub
```



## 1.5. Wie müssen Skripte im Process Engineer aussehen?

Nach all diesen vielen Vorbemerkungen nun aber zu den Möglichkeiten, die Ihnen das Skripting innerhalb des DELMIA Process Engineer bietet.

Sie müssen dazu nur noch eines grundsätzlich wissen:

Jedes Skript, das Sie schreiben, muss eine Einsprungmethode aufweisen, die als Startpunkt für die Ausführung dient. Derzeit gibt es mehrere Methoden, die als Startpunkt dienen, nämlich die Methode *"main"*, die unterschiedlichen Funktionen der Skriptaktionen und Startmethode in VBA.

„Normale“ Skripte müssen mindestens ein Gerüst der Form beinhalten.

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

Was Sie innerhalb dieses Rumpfes programmieren, bleibt Ihnen überlassen.

Sie können Ihren Code komplett innerhalb dieser Methode schreiben, ab einer gewissen Komplexität empfiehlt es sich, das Programm in Unterprogramme zu strukturieren, d.h. von der „*main*“-Methode aus weitere Untermethoden aufzurufen.

Sicher ist Ihnen in diesem Codefragment der Übergabeparameter **object\_id** aufgefallen.

### Was ist die *object\_id*?

Fast jedes Objekt, das Sie in der Baum- oder Listenansicht Ihres DELMIA Process Engineers sehen, ist persistent, das heißt dauerhaft in Ihrer Datenbank, dem PPR Hub, gespeichert und wird durch eine eindeutige Kennzahl, die Objekt-ID, identifiziert. In dem Moment, in dem Sie per Kontextmenü die Aktion „Skript ausführen“ auswählen, wird automatisch die Objekt-ID des Objektes an das Skript weitergereicht, auf welchem Sie die Aktion starten, unabhängig davon, ob Sie diese im Skript verwenden oder nicht. Die Objekt-ID ist ein String, der anhand seines Aussehens beginnend mit *"\$id\$"* erkennbar ist. Die Objekt-ID des Objektes wird deswegen übergeben, weil es in vielen Fällen notwendig ist, Attribute dieser Komponente oder zugeordnete Stücklisteneinträge, Relationen und ähnliches herauszufinden.

Beispiel für eine Befehlszeile, die die **Object\_id** nutzt:

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



**Hinweis:** In Folgeversionen des DELMIA Process Engineer wird es weitere Einsprungmethoden geben.

Neben der Hauptroutine **main** gibt es noch zwei weitere Methoden:

#### ❑ Prozedur

Eine Prozedur ist eine benannte Folge von Anweisungen, die als Einheit ausgeführt werden. Function und Sub sind zum Beispiel Prozedurtypen. Der Name einer Prozedur wird immer auf Modulebene definiert. Der gesamte ausführbare Code muss in einer Prozedur enthalten sein. Prozeduren können nicht in anderen Prozeduren eingesetzt werden, aber von anderen Prozeduren aufgerufen werden. Eine Prozedur hat keinen Rückgabewert.

## ❑ Funktionen

Der Aufbau einer Funktion ist der gleiche wie der einer Prozedur. Als Unterschied zur Prozedur hat die Funktion einen Rückgabewert.

Nun ist auch klar, warum das Eingangsbeispiel:

*MsgBox "Hallo, dies ist eine Ausgabebox", , "Mein erstes Script"*

eine Fehlermeldung gebracht hat.

Wenn Sie das Skript in der Prozedur **main** ausführen, werden Sie keine Fehlermeldung erhalten.

```
Sub main (object_id)
```

```
    MsgBox "Hallo, dies ist eine Ausgabebox", , "Mein erstes Script"
```

```
End Sub
```

## Was ist eine Variable?

Eine Variable ist eine benannte Speicherposition, die Daten enthalten kann, die während der Programmausführung verändert werden können. Jede Variable hat einen Namen, der sie eindeutig in ihrem Gültigkeitsbereich identifiziert. Zusätzlich kann ein Datentyp angegeben werden.

Variablenamen müssen mit einem Zeichen des Alphabets beginnen, innerhalb des Gültigkeitsbereichs müssen sie eindeutig sein, und sie dürfen nicht länger als 255 Zeichen sein. Variablenamen dürfen keinen Punkt und kein Typkennzeichen enthalten.

### 1.5.1.1. Fehlerbehandlung

Natürlich wäre es ein Wunder, wenn alles auf Anhieb klappen würde, aber in der Regel treten vor allem bei größeren Skripten einige Fehlermeldungen auf. Welche Fehlerarten können vorkommen?

➡ Syntaxfehler im Skript

➡ Laufzeitfehler

a) logische (typisches Beispiel: Division durch Null)

b) ungültige Parameter bei Methoden von Fremdkomponenten (z.B. File, Datenbankzugriff)

c) Ungültige Parameter bei Methoden der DELMIA Process Engineer Komponenten (data, query, grid ...)

d) bei Aufrufen unbekannter Methoden von existierenden Objekten

**Syntaxfehler** werden von der Skript-Engine vor der Ausführung des Programms erkannt und durch eine entsprechende Fehlermeldung quittiert.

**Laufzeitfehler** haben die unangenehme Eigenschaft, dass sie die Frage aufwerfen, wie mit einer solchen Situation umgegangen werden soll. Generell gilt: Wird im Skript nicht explizit die Fehlerbehandlung durchgeführt, wird das Skript nach einem Fehler ganz abgebrochen!

Das bedeutet, dass u. U. einige Daten verändert wurden und man sich nicht mehr im Ausgangszustand befindet. Ein Abbrechen ist nicht möglich, es sei denn Sie würden alle Änderungen so protokollieren, dass diese wieder „aktiv“ rückgängig gemacht werden könnten.



```

REM Skript mit Fehlerbehandlung für Zugriff auf Attribute einer Komponente.
` Der erste Aufruf erfolgt mit einem falschen Attributnamen ("namea") und führt zu ei-
ner entsprechenden Fehlermeldung.
` Der zweite Aufruf ist korrekt (Attributname "name"). Die Methode OnErrorResume-
Next verhindert den Abbruch des Skriptes bei einem Fehler!
` Das Fehlerobjekt Err wird bei einem Fehler "befüllt" und muss danach mit Err.Clear
` wieder aufgeräumt werden.
sub change_attribute(object_id, attribute_name, value)
On Error Resume Next
call data.SetAttributeById (object_id, attribute_name, value)
if Err.Number <> 0 then
    MsgBox(Err.Description)
    Err.Clear
else
    MsgBox("Ausführung korrekt")
end if
end sub

Sub main (object_id)
REM Aufruf mit falschem Attribut
call change_attribute(object_id, "namea", "Test")
REM Aufruf mit richtigem Attribut
call change_attribute(object_id, "name", "Test")
end Sub

```



**Achtung:** Als Abbruch-String für GetFirstChild/GetNextChild wird der String "" verwendet

### 1.5.1.2. Rekursionen

Sehr häufig wird es vorkommen, dass komplette Baumstrukturen durchlaufen werden müssen. Dazu können Skriptmethoden rekursiv aufgerufen werden. Man kann auch die **main**-Methode selbst rekursiv aufrufen, wie im nachfolgenden Beispiel dargestellt. Die Übersichtlichkeit solcher Skripte, vor allem bei längeren Skripten, geht jedoch verloren.

```

REM allgemein (rekursiv)
Sub main(parent_id)
    child_id = data.GetFirstChild(parent_id, "nodes")
    If child_id <> "" Then
        Do While child_id <> ""
            main(child_id)
            ...Anweisung ...
            child_id = data.GetNextChild(parent_id, "nodes")
        Loop
    End If
End Sub

```



Im Allgemeinen ist es bei solchen Aufrufen und Schleifen wichtig darauf zu achten, die Abbruchkriterien nicht zu vergessen. Sonst kommt es zu Endlosschleifen.

```
REM Kumuliere Invest (eine Ebene)
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 Kumuliere Masse (rekursiv)
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
```

Um die Liste mehrfach durchlaufen zu können, wird mit Hilfe des ResetIterator auf ScriptItemData der Iterator (also der Laufzeiger) einer bestimmten Kinderliste zurückgesetzt.

Es besteht auch die Möglichkeit, nach Zeiger-Attributen zu suchen, bzw. Zeiger-Attribute zu setzen.

Alle Komponenten vom Typ "ergocomplantdefault" des aktuellen Projektes suchen.

```
REM Suchen nach allen Ressourcen (Typ "ergocomplantdefaults") des Projekts
Sub main(id)
'project_id=data.getattributebyid(id, "ergoproject")
query.ResetSearch
query.SetQuery "ergocomplantdefault", "ergoproject", "=", project_id 'oder id
result_id=query.GetFirstResult
Do while result_id <> ""
wert = data.GetAttributebyId (result_id, "name")
MsgBox(wert)
result_id=query.GetNextResult
Loop
end Sub
```

Dieses Skript kann im Kontextmenü des Projektes aufgerufen werden, da ja die **id** des Projektes als Vergleichskriterium verwendet wird. Wenn es nicht auf das Projekt aufgerufen wird, dann muss man sich diese ID (**project\_id**) erst besorgen, z. B. **project\_id=data.getattributebyid(id, "ergoproject")**.

#### Bitte beachten Sie:

Objekte (Typen, Plantypen) können Attribute haben, die nicht über einen konfigurierten Namen (z.B. **"name"**) abgefragt werden können, weil sie in der Regel nur intern verwendet werden. Im Beispiel oben ist es jedoch notwendig an-

zugeben, dass nur innerhalb eines bestimmten Projektes gesucht werden soll. Hier muss deswegen als Attributname das Zeigerattribut **"ergoproject"** verwendet werden.



**Hinweis:** VBScript ist etwas tückisch hinsichtlich der Klammersetzung. In den Beispielen werden Sie manchmal Klammern vermissen, wenn eine Methode mehrere Parameter hat. Das ist zwingend!

### Wie gehen Sie vor:

- ➔ Hat eine Methode einen Rückgabewert, dann sind die Parameter in Klammern zu setzen:

➔ `wert = data.GetAttributebyId(child_id, "dummyinvest")`

- ➔ Hat die Methode keinen Rückgabewert also etwa:

➔ `data.SetAttributebyId parent_id, "calc_estim_invest", wert`

- ➔ dann dürfen **keine** Klammern verwendet werden.

Wem das zu undurchsichtig ist, der sollte den Befehl **call** verwenden. Damit kann die letzte Codezeile auch so geschrieben werden:

```
Call data.SetAttributebyId(parent_id, "calc_estim_invest", wert)
```

### 1.5.1.3. Weiterführende Beispiele

```
REM Write Children Names To Txt (rekursiv)

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
```

```
REM Beispiel für Verwendung eines Dictionary-Objektes
REM Legt eine Datei C:\test.txt an in der die shortnames und
REM attribute1 derjenigen Prozesse aufgelistet werden, bei de-
REM die Kombination aus beiden nicht eindeutig ist.
```

```
Sub checkProcess( parent_id, ts, d )
    wert = ""
    wert = data.GetAttributebyId(parent_id, "nameshort")
    wert2 = ""
    wert2 = data.GetAttributebyId(parent_id, "attribute_1")
    wertkomb=wert+wert2
    If (d.exists (wertkomb) ) Then
        text = "Prozess existiert bereits ts: "+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
```

### 1.5.2. Skripte in anderen Skripten verwenden

Mit der Einführung der Skripte und Skriptaktionen in der Systembibliothek ist eine Verwendung von anderen Skripten im eigentlichen Skriptcode möglich und auch beabsichtigt. Am einfachsten ist es, dieses Thema mit Hilfe eines einfachen Beispiels zu erklären:

Bei einem Datenzugriff kommt es darauf an, die richtigen Objekte aufzurufen. Dazu benutzen Sie den Aufruf:

```
base_id = Data.GetAttributeById(sci_id, "relati onobj ect2")
```

Dieser Funktionsaufruf soll leichter und wiederverwendbarer gemacht werden. Die folgenden Schritte sind erforderlich:

- ➔ Erzeugen Sie ein Skript in der Systembibliothek
- ➔ Tragen Sie einen Namen ein z. B. „BasicIncludes“
- ➔ Tragen Sie folgenden Skriptcode ein:

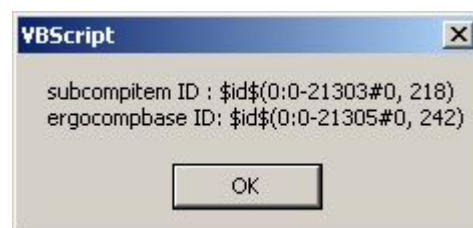
```
function GetBase(id)
  GetBase = Data.GetAttributeById(id, "relati onobj ect2")
end function
```

- ➔ Speichern Sie das Skript.
- ➔ Erzeugen Sie nun ein Skript in der Projektbibliothek.
- ➔ Geben Sie dem Skript einen Namen.
- ➔ Tragen Sie folgenden Skriptcode ein:

```
#include<BasicIncludes>

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

- ➔ Speichern Sie das Skript.
- ➔ Mit Ausnahme des Projektknotens können Sie dieses Skript nun auf jedem Knoten ausführen.
- ⇒ Als Ergebnis werden sie folgende Meldung erhalten (siehe [Abbildung 20](#)).



**Abbildung 20:** Beispiel einer Meldung bei Verwendung von Skripten in einem Skript

#### Was geschieht bei einer Verwendung eines anderen Skripts?

Die **include** Anweisung in Ihrem „main“ Skript zwingt das Programm nach einem Skript zu suchen, dessen Namen zwischen den "< >" (eckigen Klammern) steht. Die Suche beginnt innerhalb der Projektbibliothek, zu der das „main“ Skript gehört. Wenn es keine Übereinstimmung gibt, dann wird die Systembibliothek nach einem Skript durchsucht. Ist das Skript gefunden, übermittelt der Skriptinterpreter dies:

```
function GetBase(id)
  GetBase = Data.GetAttributeById(id, "relationobject2")
end function

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

### Was geschieht, wenn Skripte mit identischen Namen in der Projektbibliothek und Systembibliothek existieren?

Das Skript in der Projektbibliothek hat Priorität. Das Skript in der Systembibliothek wird ignoriert.

Jedoch können Sie den DELMIA Process Engineer zwingen, das Bibliotheksskript zu verwenden, indem Sie folgenden Code schreiben:

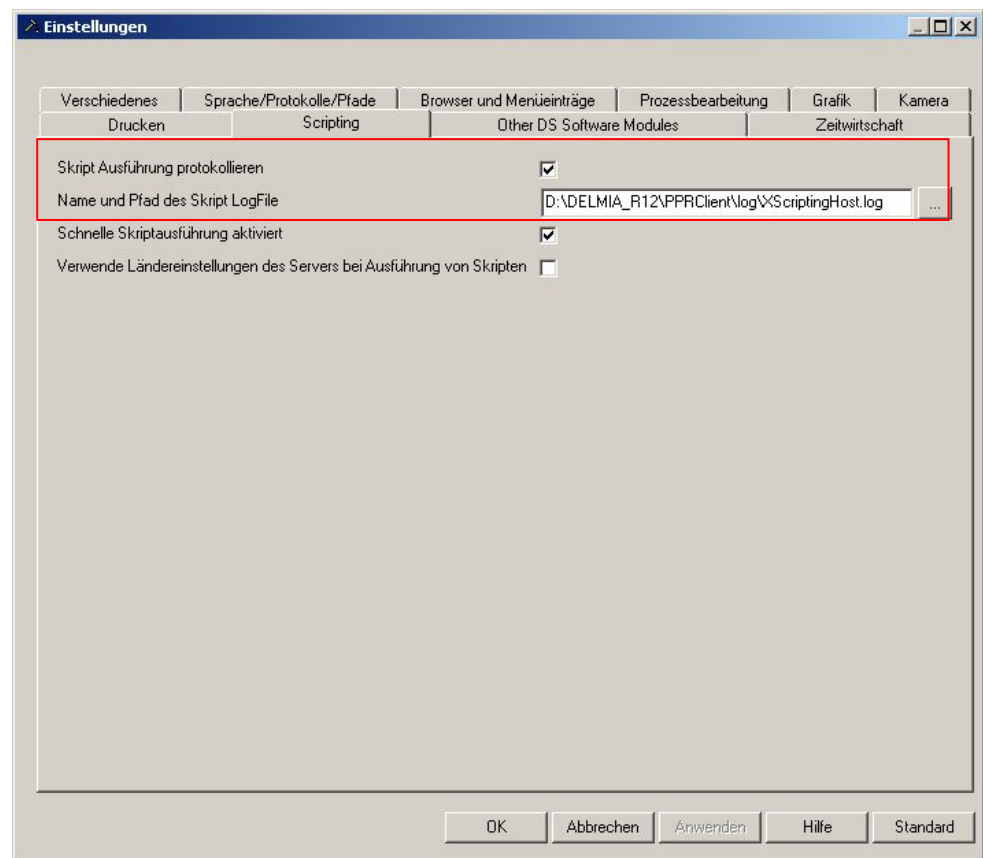
```
i ncl ude<Lib/Basic l ncl udes>
```

### Was geschieht, wenn ein Skript "per Zufall" mehrfach verwendet wird?

Nichts, jedes Skript wird tatsächlich nur einmal verwendet.

### 1.5.3. Protokollieren der Skriptaktivitäten

Über den Dialog *Einstellungen* im Menü *Werkzeuge* können Sie die Protokollfunktionen für Skripte aktivieren. Siehe auch: [Abbildung 21](#).



**Abbildung 21: Protokollfunktionen aktivieren – Werkzeuge – Einstellungen**



**Achtung:** Das Ausführen einer Skriptaktion wird durch das Aktivieren der Protokollfunktion verlangsamt.

- ➡ Um die Protokollfunktionen zu aktivieren, setzen Sie ein Häkchen bei *Skript Ausführung protokollieren*.

Ist die Protokollfunktion aktiviert, wird die Protokolldatei ständig aktualisiert. Es ist deshalb zu empfehlen, die Protokolldatei periodisch zu archivieren. Nach der Archivierung kann die Protokolldatei aus dem Verzeichnis gelöscht werden.

## 1.6. VBA-Host für die Programmierung im Process Engineer verwenden

### 1.6.1. Einleitung

Einen guten Grund mit Visual Basic Skripte (VBS) zu schreiben ist, dass diese Programmiersprache leicht zu erlernen ist und schnell in die Anwenderprogramme im Process Engineer integriert werden kann. Es ist ein guter Einstieg, um die Funktionsweise von Skripten kennen zu lernen und anzuwenden. Die Programmiersprache **VBScript** ist in ihrem Sprachumfang jedoch nicht so mächtig wie andere Programmiersprachen. Um komplexe Sachverhalte in einer eigenen Entwicklungsumgebung mittels Skripten im Process Engineer zu beschreiben, wird ab der Version PE 5.10, die Programmiersprache **Visual Basic for Applications (VBA)** verwendet.

Mit der Integration von VBA in den Process Engineer wird für das Schreiben von Skripten eine eigene komfortable Entwicklungsumgebung zur Verfügung gestellt, in der beispielsweise

- Syntaxhervorhebungen,
- Fehlerbereinigung (debuggen)
- und intelligente Funktionserweiterungen, wie etwa das automatische Aufzeigen von weiteren Programmschritten beim Schreiben von Skripten, integriert sind.

VBA-Programme laufen innerhalb der gestarteten Anwendung ab und greifen auf das Objektmodell der Anwendung zu.

Bei den folgenden Beschreibungen in diesem Kapitel wird die VBA-Umgebung als bekannt vorausgesetzt, beispielsweise aus den Office-Anwendungen. Auf die VBA-Umgebung (IDE) wird deshalb nur soweit eingegangen, wie es für das Verständnis für die einzelnen Abschnitte erforderlich werden kann.

### 1.6.2. VBA-Skripte erstellen

VBA-Projekte werden in der Projektbibliothek angelegt und erstellt. Zu den Informationen, die im Eigenschaftsdialog für VBA-Projekte eingegeben werden, wird gleichzeitig im Hintergrund ein binäres Objekt erzeugt; ein sogenanntes Binary Large Object (BLOB). In diesem binären Objekt werden alle Bezugsdaten der VBA-Projekte strukturiert gespeichert.

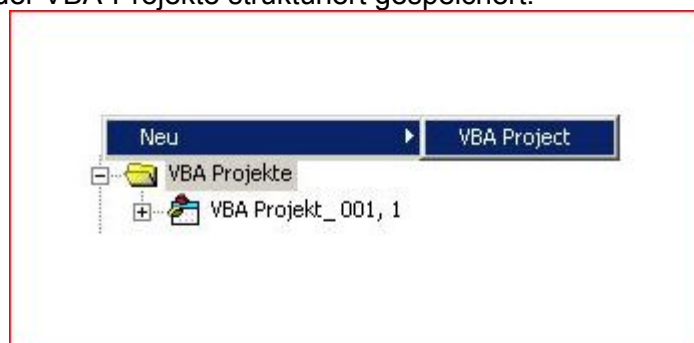


Abbildung 22: VBA-Skripte anlegen – Kontextmenü Neu

- ➡ Klappen Sie die Projektbibliothek auf.

- ➔ Selektieren Sie in der Projektbibliothek den Ordner *VBA Projekte*. Im Kontextmenü wählen Sie *VBA Project*.
- ➔ Im Eigenschaftsdialog geben Sie den Namen und Nummer für die Identifikation des Skriptes ein. Klicken Sie auf OK, um das VBA-Projekt anzulegen.

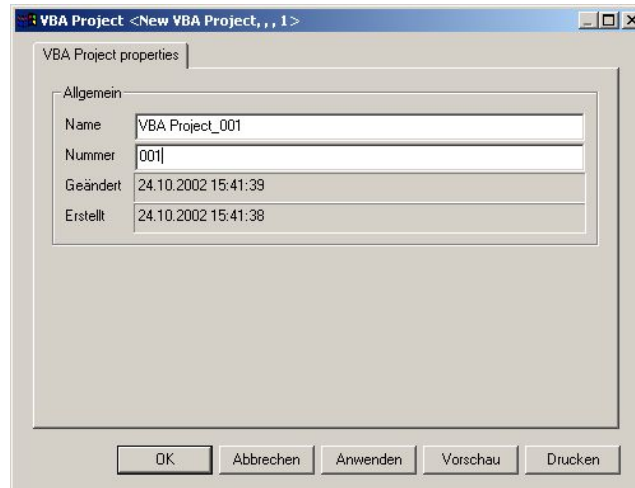


Abbildung 23: Eigenschaftsdialog für VBA-Projekte

#### 1.6.2.1. VBA-Umgebung (IDE) öffnen

In der VBA-Umgebung werden die Skripte geschrieben. Die VBA-Umgebung wird über das Kontextmenü eines angelegten VBA-Projekts in der Projektbibliothek geöffnet. In dieser Entwicklungsumgebung werden beispielsweise Makrocodes, Klassen, Module und Ressourcen entwickelt.

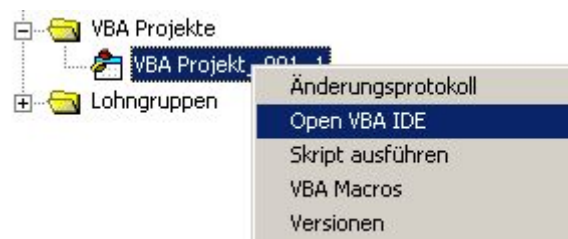


Abbildung 24: Kontextmenü – VBA-Umgebung öffnen

#### So gehen Sie vor

- ➔ Klappen Sie die Projektbibliothek auf.
- ➔ Klappen Sie in der Projektbibliothek den Ordner *VBA-Projekte* auf. Selektieren Sie danach ein *VBA-Projekt*.
- ➔ Öffnen Sie das Kontextmenü auf dem selektierten VBA-Projekt. Klicken Sie im Kontextmenü auf *Open VBA IDE*. Die VBA-Umgebung wird geöffnet.

Siehe auch: [Abbildung 25](#).

#### 1.6.2.2. Ansicht VBA-Umgebung

Das Bild zeigt beispielhaft eine VBA-Umgebung, wie sie nach dem Öffnen über das VBA-Projekt im Process Engineer angezeigt werden kann. In der



Baumstruktur des Projekts (linken Seite) werden die Methoden für das Skript-erstellen (XscriptItem-Objekte) angezeigt, die Ihnen aus dem Process Engineer vertraut sein sollten.



- ➔ Klicken Sie auf dieses Icon, die Entwicklungsumgebung wird verlassen und Sie befinden sich wieder im Process Engineer.

## Beispiel

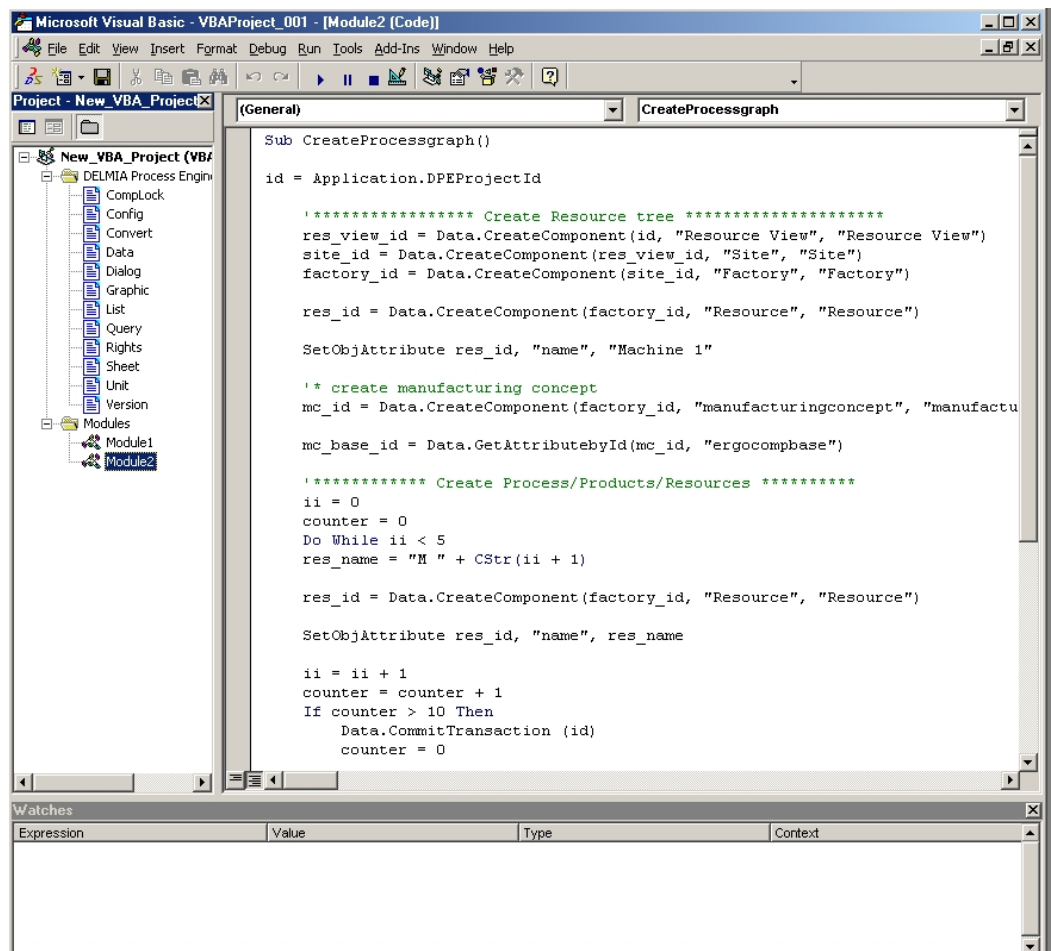


Abbildung 25: Beispiel für eine VBA-Umgebung

### 1.6.3. VBA-Makros ausführen

VBA-Makros werden in der Projektbibliothek ausgeführt. Über den Button *Create* können Sie in der VBA-Umgebung ein neues Makro erzeugen und danach auch ausführen. Über den Button *Edit* können Sie erzeugte Makros bearbeiten.



Abbildung 26: VBA-Makros über Kontextmenü öffnen

#### So gehen Sie vor

- ➔ Klappen Sie die Projektbibliothek auf.
- ➔ Klappen Sie in der Projektbibliothek den Ordner *VBA-Projekte* auf. Selektieren Sie danach ein *VBA-Projekt*.
- ➔ Öffnen Sie das Kontextmenü auf dem selektierten VBA-Projekt. Klicken Sie im Kontextmenü auf *VBA Macros*.
- ➔ Im Dialog **Macros** können angelegte Macros unter **Macros in** projektbezogen und projektweit für die Ausführung ausgewählt werden.

Siehe auch: [Abbildung 27](#).

#### Beispiel

Macros entweder projektweit oder projektbezogen auswählen.

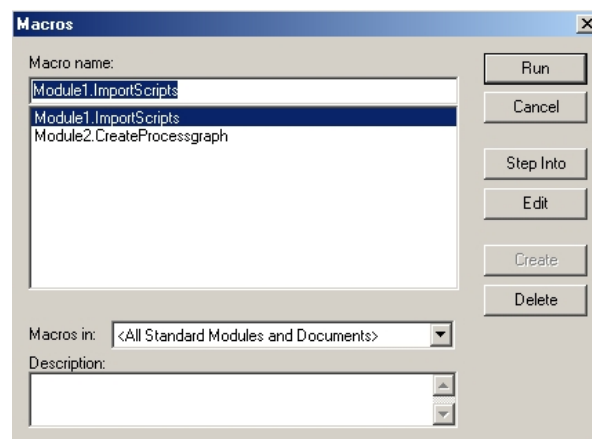
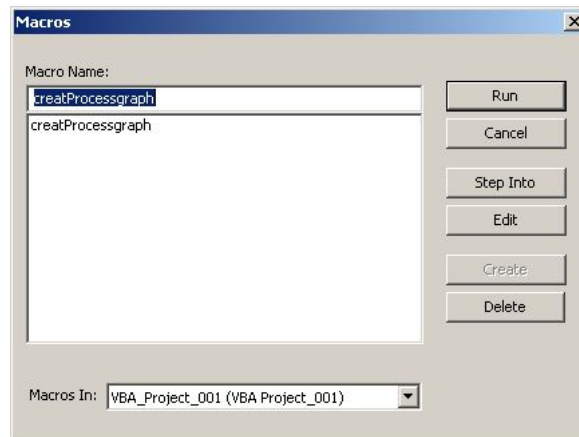


Abbildung 27: Beispiel - Dialog Macros – projektweite Auswahl

- ➔ Um ein Makro auszuführen, selektieren Sie ein Makro und klicken danach auf den Button *Run*. Das Makro wird ausgeführt.
- Für eine projektbezogene Auswahl, siehe auch: [Abbildung 28](#).

Beispiel für eine projektbezogene Auswahl

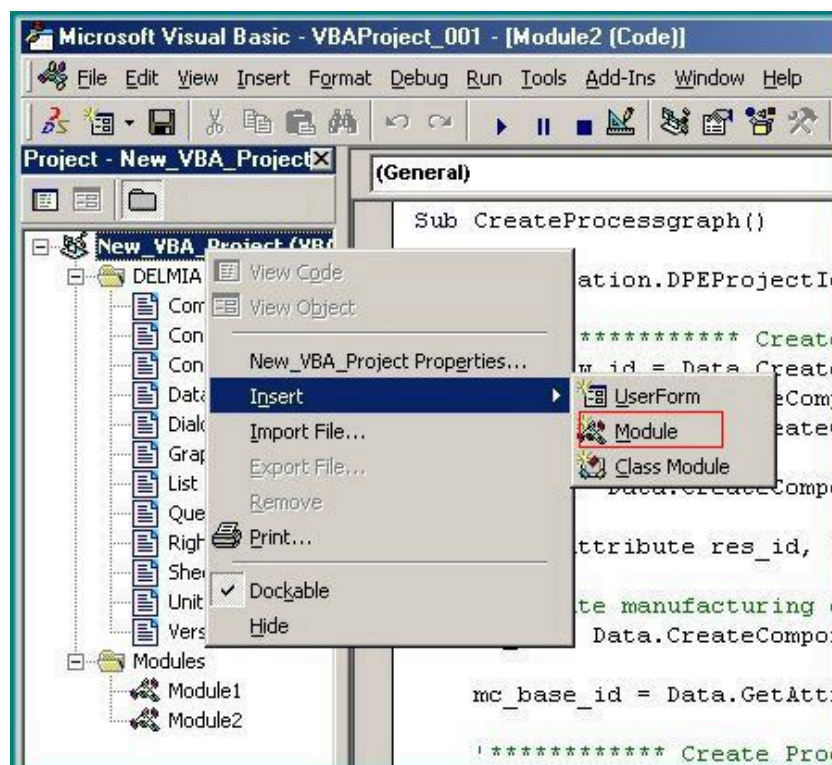
#### Beispiel



**Abbildung 28: Beispiel - Dialog Macros – projektbezogene Auswahl**

### 1.6.4. Skripte in VBA-Makros umwandeln

VBA-Skripte werden in der VBA-Umgebung bearbeitet. Um einen vorhandenen Schriftcode zu bearbeiten bzw. umzuwandeln, muss ein VBA-Projekt erzeugt worden sein und die VBA-Umgebung geöffnet werden. Am nachfolgenden Beispiel wird die Vorgehensweise erklärt.



**Abbildung 29: Skripteditor öffnen**

- ➔ Selektieren Sie in der Baumstruktur das Projekt (im Bild wird ein neu angelegtes Projekt gezeigt, siehe auch: [Abbildung 29](#)).
- ➔ Öffnen Sie danach das Kontextmenü, und klicken Sie beispielweise auf *Module*.
- ➔ Schreiben Sie im Skripteditor das Skript.

- ➡ Damit das Skript als Makro erkannt wird, müssen Sie die Einträge gemäß dem im Beispiel gezeigten Aufbau schreiben.

Im rechten Fenster wird der Aufbau (blaue Schrift) für die Umwandlung des Skripts in ein VBA-Makro gezeigt.

Siehe auch: [Abbildung 30](#).

### Beispiel

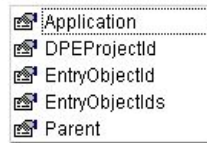
REM Script code	REM VBA macro code
sub main( <b>entry_id</b> )	sub DisplayName()
name = Data.GetAttributeById(entry_id, "name")	<b>entry_id = Application.EntryObjectId</b>
MsgBox(name)	name = Data.GetAttributeById(entry_id, "name")
end sub	MsgBox(name)
	end sub

**Abbildung 30: Beispiel für Skript in Makro umwandeln**

#### 1.6.4.1. VBA Objekt Application

Jedes Mal, wenn Sie das Wort "Application" gefolgt von einem einzelnen Punkt schreiben, schlägt die VBA IDE eine Eigenschaft des VBA *Application* Objektes zur Auswahl vor.

```
ID = Application.
```



Die Erläuterung der Eigenschaften finden Sie in der nachfolgenden Tabelle:  
Siehe auch: [Abbildung 31](#).

Entsprechend den in der Tabelle erläuterten Begriffen werden diese Eigenschaften dynamisch mit der Objektkennung einem Knoten zugeteilt, auf dem das VBA Makro ausgeführt wird.

Eigenschaften VBA Objekten	Erläuterung	
EntryObjectId	ID des selektierten Objekts	
DPEProjectId	ID des Projekts, zudem das Objekt gehört, (der zurzeit laufenden Transaktion).	
EntryObjectIds	Objekt ist vom Basistyp.	Inhalt des Felds
	subcompitem	EntryObjectIds(0) = ID of subcompitem EntryObjectIds(1) = ID of referenced ergocompbase
	relationship	EntryObjectIds(0) = ID of relationship EntryObjectIds(1) = ID of referenced relationobject
	andere	EntryObjectIds(0) = object ID EntryObjectIds(1) = object ID

**Abbildung 31: Erläuterung der Eigenschaften von VBA Objekten**

### 1.6.5. Fehlerbereinigung von VBA-Makros.

Eines der wichtigsten Merkmale der VBA-Umgebung (IDE) ist der integrierte Debugger. Jemand, der häufig für die Entwicklung von Makrocodes verantwortlich ist, weiß diese Art der Unterstützung zu schätzen. Die Eingangsobjektkennung betreffend müssen Sie die folgende Tatsache beachten: Wie erwähnt, ist die EntryObjectID die ID des selektierten Objekts. Wenn Sie in der VBA-Umgebung (IDE) Ihr Skript zu Testzwecken ausführen, werden Sie immer die Entry ObjectID des VBA-Projektes erhalten. Aber diese entspricht fast nie der benötigten ID.

#### Beispiel

```
REM VBA Makro Code
REM Richtige Rückgabe der EntryObjectID
sub Displayname()
    entry_id = "$id$(0: 0-9201#0, 218)"
    name = Data.GetAttributebyId(entry_id, "name")
    MsgBox(name)
end sub
```

Das Skript gibt die ID desjenigen Objektknotens im PPR-Navigator zurück, auf welchem es ausgeführt wurde.

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

## 2. Liste aller Skriptaktionen

Skript-Aktionen	Skript-Funktion zum Einbinden	Ereignis	Überprüft wird	Vor	Nach
<b>Skriptaktion Neues Kind</b>	sa_newchild	Anlegen einer neuen Komponente	Parent object	<b>X</b>	-
<b>Skriptaktion Kind hinzufügen</b>	sa_addchild	Hinzufügen einer Komponente zur Kindliste.	Parent object	-	<b>X</b>
<b>Skriptaktion Attribute setzen</b>	sa_setattribute	Ändern von Attributen einer Komponente.	Object	<b>X</b>	<b>X</b>
<b>Skriptaktion Neu</b>	sa_new	Nachdem eine neue Komponente angelegt wurde.	Object	-	<b>X</b>
<b>Kopieren</b>	sa_copy	Kopieren von Komponenten (einzelne und Mehrfachauswahl) zu einem anderen Vaterobjekt.	Parent object	<b>X</b>	<b>X</b>
<b>Verknüpfen</b>	sa_link	Verlinken von Komponenten (einzelne und Mehrfachauswahl) zu einem anderen Vaterobjekt.	Parent object	<b>X</b>	<b>X</b>
<b>Verschieben</b>	sa_move	Verschieben von Komponenten (einzelne und Mehrfachauswahl) von einem zu einem anderen Vaterobjekt.	Parent object	<b>X</b>	<b>X</b>
<b>Löschen</b>	sa_delete	Löschen einer Komponente (aus der Projektbibliothek).	Object	<b>X</b>	-
<b>Kind entfernen</b>	sa_removechild	Löschen einer Verlinkung (subcompitem, relationship)	Parent object	<b>X</b>	<b>X</b>
<b>Eigenschaftsdialog öffnen</b> <b>Skriptaktion Init Properties)</b>	sa_initproperties	Nach der Initialisierung eines Eigenschaften Dialogs (aber vor dem tatsächlichen Öffnen des Dialogs)	Object	-	<b>X</b>
<b>Change Properties</b>	sa_changeproperties	Nach dem Ändern der Werte einer Combobox oder einer Listbox in einem Eigenschaften Dialog	Object	-	<b>X</b>

Skript-Aktionen	Skript-Funktion zum Einbinden	Ereignis	Überprüft wird	Vor	Nach
<b>Neue Version</b>	sa_newversion	Erstellen einer neuen Version oder Versionsprüfung .	Object	<b>X</b>	<b>X</b>
<b>SetPlanningState</b>	sa_setplanningstate	Setzen des Planungsstatus oder Überprüfen einer Komponente.	Object	<b>X</b>	<b>X</b>
<b>OpenProject</b>	sa_openproject	Das zugehörige Skript wird jedes Mal ausgeführt, nachdem ein Projekt geöffnet wurde.	Object	-	<b>X</b>
<b>CloseProject</b>	sa_closeproject	Das zugehörige Skript wird jedes Mal ausgeführt, bevor ein Projekt geschlossen wird.	Object	<b>X</b>	-
<b>SelectProject</b>	sa_selectproject	Zugriff auf den Projekt öffnen Dialog.	Object	-	<b>X</b>
<b>Print</b>	sa_print	Bevor und nachdem ein Druckauftrag gestartet wurde	Object	<b>X</b>	<b>X</b>
<b>PrintList</b>	sa_printlist	Bevor und nachdem ein Listendruckauftrag gestartet wurde	Object	<b>X</b>	<b>X</b>
<b>StartBalancing</b>	sa_bal_start	Das zugehörige Skript wird jedes Mal ausgeführt, nachdem eine Austaktung geöffnet wurde.	Object	-	<b>X</b>
<b>SaveBalancing</b>	sa_bal_end	Das zugehörige Skript wird jedes Mal nach dem Speichern einer Austaktung ausgeführt.	Object	-	<b>X</b>
<b>DropProcessToBalancing</b>	sa_bal_drop_proc	Das zugehörige Skript wird nach dem Einfügen eines Prozesses (Drag & Drop) in die Austaktung ausgeführt.	Object	-	<b>X</b>
<b>Opengraphic</b>	sa_opengraphic	Vor dem Öffnen oder Anzeigen eines Grafikobjektes.	Object	<b>X</b>	-
<b>Allgemein</b>	Abhängig von den Parametern	Spezielle, kundenangepasste Anwendung		<b>X</b>	<b>X</b>



## 2.1. Skriptaktion Neues Kind

### Beschreibung

Das zugehörige Skript wird jedes Mal ausgeführt, wenn Sie ein neues Kind über den Kontextmenüeintrag „Neu“ hinzufügen. Das Skript kann das Ereignis durch Zurückgeben von 0, false = VbFalse (VBScript) oder false (JScript), beenden.

### Funktion zum Starten der Skriptaktion

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

Parameter	Beschreibung
<code>parent_id</code>	Ein String, der die ID des Vaterobjektes enthält.
<code>childname</code>	<ul style="list-style-type: none"> <li>bis PE 5.13: Ein String, der den Namen des Planungstypen oder den Typennamen der Komponente enthält, die erstellt oder hinzugefügt werden soll.</li> <li>ab PE 5.14 and später: Ein String, der die GUID des Planungstypen oder den Typennamen der Komponente enthält, die erstellt oder hinzugefügt werden soll.</li> </ul>
<code>childlistname</code>	<p>Ein String, der den Namen der Kinderliste enthält, in die das Kind hinzugefügt werden soll.</p> <ul style="list-style-type: none"> <li>bis PE 5.13: Für PPR components "nodes" oder der Name des Planungstyps, für andere Relationen (relationships) der konfigurierte ,childlistname'.</li> <li>ab PE 5.14 and später: Für PPR components entweder leer (empty) oder "nodes", für andere Relationen (relationships) der konfigurierte ,childlistname'.</li> </ul>

### Beispiel

#### Beispiel einer Skriptaktion

```
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
```

## 2.2. Skriptaktion Kind hinzufügen

### Beschreibung

Das zugehörige Skript wird jedes Mal ausgeführt, wenn Sie ein neues Kind über den Kontextmenüeintrag „Neu“ hinzufügen.

### Dazugehörige Funktion

*function sa\_addchild(parent\_id, child\_id, childlistname)*

Parameter	Beschreibung
parent_id	Ein String, der die ID des Vater Objekts enthält.
child_id	Ein String, der die ID des Kind Objekts enthält, das hinzugefügt wird.
childlistname	<p>Ein String, der den Namen der Kindliste enthält, in die das Kind hinzugefügt wurde.</p> <ul style="list-style-type: none"> <li>• bis PE 5.13: Für PPR-Komponenten "nodes" oder der Name des Planungstyps. Für andere Relationen (relationships) der konfigurierten ,childlistname'.</li> <li>• ab PE 5.14: Ein String, der die GUID des Planungstypen oder den Typennamen der Komponente enthält, die hinzugefügt werden soll.</li> <li>• PE 5.15 und später: Für PPR-Komponenten: "nodes" oder der Name des Planungstyps, für andere Relationen (relationships) der konfigurierten ,childlistname'.</li> </ul>

### Beispiel

#### Beispiel einer Skriptaktion

```
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
```

## 2.3. Skriptaktion Attribute setzen

### Beschreibung

Das zugehörige Skript wird jedes Mal ausgeführt, wenn Sie Attribute in den Eigenschaften-Dialog ändern. Die Skriptaktion benötigt die Object\_ID und eine Liste aller geänderten Attributwertepaare. Das Skript kann das (Vor-) Ereignis durch Zurückgeben von 0 oder falsch beenden. Wenn dies geschieht, bleibt der Eigenschaften Dialog geöffnet. Die alten Attributwerte werden nicht wiederhergestellt.

### Dazugehörige Funktion

*function sa\_setattribute(object\_id, attributes)*

Parameter	Beschreibung
<i>object_id</i>	Ein String, der die ID des Objektes enthält, dessen Attribute geändert werden (geändert wurden).
<i>attributes</i>	Ein Feld (Liste), das alle Attribute enthält, die der Benutzer im Eigenschaftsdialog geändert hat.

### Beispiel

#### Beispiel einer Skriptaktion

```
function sa_setattribute(id, values)  
dim mx = 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  
else  
    sa_setattribute = False  
end if  
end function
```

Der Inhalt der Liste der geänderten Attribute unterscheidet sich für SA\_BEFORE und SA\_AFTER.

- Im Falle von SA\_AFTER werden die Werte (nicht die Namen der Attribute) auf Null gesetzt (wenn das Setzen aller Attribute erfolgreich war). Dies ist ein Betriebssystemmerkmal, weil die zurückgegebene Null eine Erfolgsmeldung bedeutet (ähnlich S\_OK).

Hintergrund dieser Funktion: Wenn nur ein Attribut im Feld falsch ist oder nicht geschrieben werden kann, dann enthält der zurückgegebene Wert für dieses Attribut zusätzliche Fehlerinformationen. Diese Information kann dann vom Skript weiterverarbeitet werden.

## 2.4. Skriptaktion Neu

### Beschreibung

Das zugehörige Skript wird jedes Mal ausgeführt, wenn Sie eine neue Komponente erzeugen.

### Dazugehörige Funktion

**Alternative A** (alle Versionen): `function sa_new(object_id)`

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

Parameter	Beschreibung
<code>object_id</code>	Ein String, der die ID des neuen Objekts enthält
<code>link_id</code>	Ein String, der die ID des neuen Objekts enthält. Dies ist die ID eines Stücklisteneintrages (BOM entry) wenn das neue Objekt eine PPR Komponente ist. Ansonsten ist sie mit der <code>object_id</code> identisch.

### Beispiel

#### Beispiel einer Skriptaktion

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

## 2.5. Skriptaktion Kopieren

### Beschreibung

Das zugehörige Skript wird jedes Mal ausgeführt, wenn Sie eine neue oder mehrere Komponenten kopieren. Das Skript kann das Ereignis durch Zurückgeben von 0, false = VbFalse (VBScript) oder false (JScript) beenden.

### Dazugehörige Funktion

*function sa\_copy(parent\_id, children\_ids, childlistname, children\_link\_ids)*

Parameter	Beschreibung	
<code>parent_id</code>	Ein String, der die ID des Vater-Objekts enthält.	
<code>children_ids</code>	Ein Feld, das die IDs der zu kopierenden Komponenten (vorher) oder die IDs der kopierten Komponenten (danach), enthält	
	Ein Feld, das die IDs der zu kopierenden Komponenten ( <b>base</b> objects) enthält.	Ein Feld, das die IDs der kopierten Komponenten ( <b>link</b> objects), enthält.
<code>childlistname</code>	<ul style="list-style-type: none"> <li>bis PE5.13SP6 / PE5.14SP2: Ein String, der entweder "nodes, den Namen des Planungstypen oder den Namen des Typen (relationship) der Kinderliste enthält, in die neuen Kinder kopiert werden (worden sind).</li> <li>ab PE5.13SP7/ PE5.14SP3: Ein String, der entweder die GUID des Planungstypen oder den Namen des Typen (relationship) der Kinderliste enthält, in die neuen Kinder kopiert werden (worden sind).</li> </ul>	
<code>children_link_ids</code>	Ein Feld von Strings, das die ID der zu kopierenden Komponenten ( <b>link</b> objects) enthält.	

### Beispiel

#### Beispiel einer Skriptaktion

```
function sa_copy(parent_id, children_ids, childlistname)
parentname = data.GetAttributeById(parent_id, "name")
dim x = UBound(children_ids, 1)
For i = 0 To dim x
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
```

#### Alternativer Funktionsaufruf

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

## 2.6. Skriptaktion Verknüpfen

### Beschreibung

Das zugehörige Skript wird jedes Mal ausgeführt, wenn Sie ein oder mehrere Bestandteile mit einem anderen Elternobjekt verlinken. Das Skript kann das Ereignis durch Zurückgeben von 0, false = VbFalse (VBScript) oder false (JScript) beenden.

### Dazugehörige Funktion

*function sa\_link(parent\_id, children\_ids, childlistname, children\_link\_ids)*

Parameter	Beschreibung	
	SA_BEFORE	SA_AFTER
<i>children_ids</i>	Ein Feld von Strings, das die IDs der zu verlinkenden Komponenten ( <b>base</b> objects) enthält	Ein Feld von Strings, das die IDs der verlinkten Komponenten ( <b>link</b> objects) enthält
<i>childlistname</i>	<ul style="list-style-type: none"> <li>ab PE5.13SP6/PE5.14SP2: Ein String, der entweder den Namen des Planungstypen oder "nodes", oder den Namen des Typen (relationship) der Kinderliste enthält in die neuen Kinder verlinkt werden (worden sind).</li> <li>ab PE5.13SP7/ PE5.14SP3 und später: Ein String, der entweder die GUID des Planungstypen oder den Namen des Typs (relationship) in die neuen Kinder verlinkt werden (worden sind).</li> </ul>	
<i>children_link_ids</i>	Ein Feld von Strings, das die IDs der zu verlinkenden Komponenten ( <b>link</b> objects) enthält	

### Beispiel

#### Beispiel einer Skriptaktion

```
function sa_link(parent_id, children_ids, childlistname)
  parentname=data.GetAttributeById(parent_id, "name")
  dim x = ubound(children_ids, 1)
  for i = 0 to dim x
    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
```

## 2.7. Skriptaktion Verschieben

### Beschreibung

Das zugehörige Skript wird jedes Mal ausgeführt, wenn Sie ein oder mehrere Bestandteile von einem Elternobjekt zu einem anderen bewegen. Das Skript kann das Ereignis durch Zurückgeben von 0, false = VbFalse (VBScript) oder false (JScript) beenden.

### Dazugehörige Funktion

*function **sa\_move**(oldparent\_id, newparent\_id, children\_ids, childlistname)*

Parameter	Beschreibung	
	SA_BEFORE	SA_AFTER
oldparent_id	Ein String, der die ID des alten Vaterobjekts enthält.	Ein String, der die ID des alten Vaterobjekts (bis PE5.13) oder die ID des neuen Vaterobjekts (PE5.14 und später) enthält.
newparent_id	Ein String, der die ID des neuen Vaterobjekts enthält.	
children_ids	Ein Feld von Strings, das die IDs der zu verschiebenden Komponenten enthält ( <b>link</b> objects).	Ein Feld von Strings, das die IDs der verschobenen Komponenten enthält ( <b>link</b> objects).
childlistname	Ein String, der den Namen der Kinderliste enthält, in die neuen Kinder hinzugefügt werden (worden sind).	

### Beispiel

#### Beispiel einer Skriptaktion

```
function sa_move(oldparent_id, newparent_id, children_ids, childlistname)
oldparent_name=data.GetAttributebyId(oldparent_id, "name")
newparent_name=data.GetAttributebyId(newparent_id, "name")
dim x = ubound(children_ids, 1)
for i = 0 to dim x
    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
```

## 2.8. Skriptaktion Löschen

### Beschreibung

Das zugehörige Skript wird jedes Mal ausgeführt, wenn Sie eine Komponente löschen. Das Skript kann das Ereignis durch Zurückgeben von 0, false = VbFalse (VBScript) oder false (JScript) beenden.



**Hinweis:** Wenn Sie diese Skriptaktion auf einem subcompitem oder einem relationship Objekt aufrufen, wird diese Skriptaktion nur dann ausgeführt, wenn auch das referenzierte Objekt der Projektbibliothek gelöscht wird. Sonst wird die Skriptaktion RemoveChild aufgerufen.

### Dazugehörige Funktion

function *sa\_delete(object\_id)*

Parameter	Beschreibung
<i>object_id</i>	Ein String, das die ID des Objekts enthält, das gelöscht wird.

### Beispiel einer Skriptaktion

```
function sa_delete(object_id)
  name = Data.GetAttributeById(object_id, "name")
  text = "This is 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
```



**Hinweis:** Ab der Version PE5.15SP3 und den nachfolgende Versionen kann die Skriptaktion Löschen auch **nach** einem Löschen des Objektes aufgerufen werden. Beachten Sie bitte, dass die ID des gelöschten Objektes ‚empty‘ ist, da das Objekt nicht mehr existiert.

Wollen Sie spezielle Informationen von dem gelöschten Objekt weiterverwenden, empfiehlt es sich diese Informationen, temporär mittels einer Skriptaktion, **VOR** dem Löschen zu speichern, z. B. in einer Textdatei. Mit der Skriptaktion DANACH können Sie diese Informationen weiterverarbeiten



## 2.9. Skriptaktion Kind entfernen

### Beschreibung

Das zugehörige Skript wird jedes Mal ausgeführt, wenn Sie eine verknüpfte Komponente (subcompitem, relationship) löschen. Wenn Sie die Komponente selbst aus der Projektbibliothek löschen, dann wird die Skriptaktion Löschen aufgerufen. Das Skript kann das Ereignis durch Zurückgeben von 0, false = VbFalse (VBScript) oder false (JScript) beenden.

### Dazugehörige Funktion

#### Alternative A

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

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

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

Parameter	Beschreibung	
<i>parent_id</i>	Ein String, der die ID des Vaterobjekts enthält.	
<i>child_id</i>	Ein String, der die Objekt- ID der Komponente enthält, die gelöscht wird.	
<i>childlistname</i>	<ul style="list-style-type: none"> <li>Für Stücklisteneinträge: "nodes".</li> <li>Ansonsten: Typenname oder Relationship_Name</li> </ul>	
<i>child_usage_id</i> (PE5.14SP2 and later)	SA_BEFORE	SA_AFTER
	Ein String, der die Objekt ID der Komponente enthält, die gelöscht wird.. Für PPR Komponenten ist dies entweder ein Stücklisteneintrag oder ein Relationsobjekt.	Identisch mit child_id

### Beispiel einer Skriptaktion

#### Beispiel

```
function sa_removechild(parent_id, child_id, childlistname)
  namep = data.GetAttributebyId(parent_id, "name")
  namec = data.GetAttributebyId(child_id, "name")
  text = "This is your last chance. You are going to remove the object
<" + 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
```

## 2.10. Skriptaktion Init Properties

### Beschreibung

Das zugehörige Skript wird jedes Mal ausgeführt, wenn der Eigenschaften-Dialog geöffnet wird. Die Skriptaktion wird **nach** (After) der Initialisierung der Daten aus der Datenbank ausgeführt, aber **bevor** der Dialog geöffnet wird. Auf diese Art ist es möglich, Werte von bestimmten Attributen zu überschreiben, Werte in Comboboxen oder Listboxen anzuhängen, zu entfernen oder zu ersetzen.

Vom Skript werden die Objekt-IDs der Komponenten übergeben, deren Attribute im Dialog gezeigt werden. Näheres in der Parameterbeschreibung.

### Dazugehörige Funktion

function *sa\_initproperties(object\_ids)*

Eigenschaften Dialog für ...	Objekt IDs
BOM entries (Stücklisteneintrag)	1. Die ID des Basisobjektes, worauf der BOM-Eintrag verweist. 2. Die ID des BOM(Stücklisten)-Eintrag (= Verwendungsdaten).
Relationship objects	1. Die ID der ersten verwendeten Komponente. 2. ID der zweiten verwendeten Komponente. 3. ID des Relationshipobjektes (Verwendungsdaten).
Andere Objekte	1. Die ID des Objekts

### Beispiel einer Skriptaktion

#### Beispiel

```
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

```

Um diese Skriptaktion verwenden zu können, benötigen Sie die Funktionen:

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

## 2.11. Skriptaktion Change Properties

### Beschreibung

Das zugehörige Skript wird jedes Mal ausgeführt, wenn der Wert eines Attributes mit dem Controltyp:

- **Combobox**
- **Listbox**
- **Bearbeiten,**
- **Multiline Edit**
- **Radiobutton**
- **Checkbox**
- **date control**

Standard  
Nur Bezeichnung  
Combobox  
Bearbeiten  
Checkbox  
Radiobutton  
Listbox  
Multiline Edit  
Browserliste  
Dateiauswahl  
RTF bearbeiten  
Ordnerauswahl  
File Viewer  
Script Editor

im Eigenschaftendialog geändert wurde. Auf diese Art ist es möglich, Werte in anzuhängen, zu entfernen oder zu ersetzen.

An das Skript werden die Objekt IDs der Komponenten und die Attribut ID (= Name) übergeben. Näheres in der Parameterbeschreibung.

### Dazugehörige Funktion

*function sa\_changeproperties(object\_id, attribute\_id)*

Parameter	Beschreibung
<code>object_id</code>	Die ID des betroffenen Objektes.
<code>attribute_id</code>	Der Name des Attributes, das geändert worden ist.

### Beispiel einer Skriptaktion

#### Beispiel

```
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 Dialog.PropSetValues(object_id, "carbodyposition", use_external, Values)
    End if
    If classification = "c1" then ' set carbodyposition menu for standard 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 Dialog.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
```

## 2.12. Skriptaktionen Neue Version

### Beschreibung

Das zugehörige Skript wird jedes Mal ausgeführt, wenn Sie eine neue Version (vor) erzeugen oder eine neue Version (nach) erzeugt wurde. Das Skript kann das Ereignis durch Zurückgeben von 0, *false* = VbFalse (VBScript) oder *false* (JScript) beenden.

### Dazugehörige Funktion

*function sa\_newversion(object\_id)*

Parameter	Beschreibung
<i>object_id</i>	Eine Zeichenfolge, die die eindeutige Kennung (ID) des Objektes enthält, wovon eine neue Version erstellt werden soll (erstellt wurde).

### Beispiel einer Skriptaktion

#### Beispiel

```
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 [" + ob-
  ject_name + "] currently having planning state [" + ps_name + "]. Con-
  tinue?"
  retval = Dialog.MessageBoxExt("Script Warning", text, "YES_NO")
  if retval = 6 then
    sa_newversion= True
  else
    sa_newversion = False
  end if
end function
```

## 2.13. Skriptaktion Planungsstatus setzen

### Beschreibung

Das zugehörige Skript wird jedes Mal ausgeführt, wenn Sie einen neuen Planungsstatus (vor) erzeugen oder ein neuer Planungsstatus (nach) erzeugt wurde. Das Skript kann das Ereignis durch Zurückgeben von 0, false = VbFalse (VBScript) oder false (JScript) beenden.

### Dazugehörige Funktion

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

Parameter	Beschreibung
<code>object_id</code>	Ein String, der die ID des Objekts enthält, dessen Planungsstatus geändert werden soll (geändert wurde)
<code>planningstate_id</code>	Ein String, der die ID des zu setzenden Planstatus enthält.

### Beispiel einer Skriptaktion

#### Beispiel

```
function sa_setplanningstate(object_id, planningstate_id)
object_name=Data.GetAttributeById(object_id, "name")
ps_name=Data.GetAttributeById(planningstate_id, "name")
text = "Sie haben vor, den Planungsstatus des Objektes [" + object_name + "] mit [" + ps_name + "] zu ersetzen. Wollen Sie fortfahren?"
retval = Dialog.MessageBoxExt("Skript Warnung", text, "YES_NO")
if retval = 6 then
sa_setplanningstate= True
else
sa_setplanningstate = False
end if
end function
```

## 2.14. Skriptaktion OpenProject

### Beschreibung

Das zugehörige Skript wird jedes Mal ausgeführt, nachdem ein Projekt geöffnet wurde.

Sie müssen den Typ "ergoproject" selektieren.

### Dazugehörige Funktion

*function sa\_openproject(object\_id)*

Parameter	Beschreibung
<i>object_id</i>	Ein String, der die ID des zu öffnenden Projektes enthält.

### Beispiel einer Skriptaktion

#### Beispiel

```
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
```



## 2.15. Skriptaktion CloseProject

### Beschreibung

Das zugehörige Skript wird jedes Mal ausgeführt, bevor ein Projekt geschlossen wird.

Das Skript kann das Ereignis durch Zurückgeben von 0, false = VbFalse (VBScript) oder false (JScript) beenden.

Sie müssen den Typ "ergoproject" selektieren.

### Dazugehörige Funktion

*function sa\_closeproject(object\_id)*

Parameter	Beschreibung
<i>object_id</i>	Ein String, der die ID des zu schließenden Projektes enthält.

### Beispiel einer Skriptaktion

#### Beispiel

```
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
```

## 2.16. ScriptAction SelectProject

### Beschreibung

Das zugehörige Script wird ausgeführt nachdem ein Projekt ausgewählt wurde.



---

### Hinweis:

*Sie müssen den Typ "ergoproject" für diese Skriptaktion auswählen.*

---

### Dazugehörige Funktion

`function sa_selectproject(object_id)`

Parameter	Beschreibung
<code>object_id</code>	Ein String, der die ID des ausgewählten Projektes enthält.

## 2.17. Skriptaktion StartBalancing

### Beschreibung

Das zugehörige Skript wird jedes Mal ausgeführt, wenn die Austaktung geöffnet wird.

### Dazugehörige Funktion

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

Parameter	Beschreibung
<code>resource_id</code>	Ein String, der die ID der Ressourcen-ID (Eltern-Objekt) enthält.
<code>process_ids</code>	Ein Feld von Strings, das die IDs der in der Austaktung eingefügten Prozesse enthält.

### Beispiel einer Skriptaktion

#### Beispiel

```
function sa_bal_start(resource_id, process_ids)
    name = Data.GetAttributeById(resource_id, "name")
    MsgBox("WLB öffnen: " + vbCrLf + "Anlage Name (Resource Name): " + vbCrLf + name )

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



## 2.18. Skriptaktion SaveBalancing

### Beschreibung

Das zugehörige Skript wird jedes Mal ausgeführt, wenn die Austaktung gespeichert wird.

Berücksichtigt wird die ID der Ressource (Eltern-Objekt), die gelöschten Prozess-IDs und die neuen Prozess-IDs.

### Dazugehörige Funktion

*function sa\_bal\_end(resource\_id, deleted\_process\_ids, new\_process\_ids)*

Parameter	Beschreibung
<i>parent_id</i>	Ein String, der die ID der Ressourcen (Eltern-Objekt) enthält.
<i>deleted_process_ids</i>	Ein Feld von Strings, welches die Ids (=ID der Relation) der Prozesse enthält, die seit dem Öffnen oder Speichern der Austaktung gelöscht worden sind.
<i>new_process_ids</i>	Ein Feld von Strings, das die IDs (=ID der Relation) der Prozesse enthält, die seit dem Öffnen oder Speichern der Austaktung dem Austaktungsblatt hinzugefügt (neue Prozesse) worden sind.

### Beispiel einer Skriptaktion

#### Beispiel

```
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
```

## 2.19. Skriptaktion DropProcessToBalancing

### Beschreibung

Das zugehörige Skript wird jedes Mal ausgeführt, wenn ein Prozess in das Austaktungsblatt eingefügt wird.

### Dazugehörige Funktion

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

Parameter	Beschreibung
<code>resource_id</code>	Ein String, der die ID der Ressourcen (Eltern-Objekt) enthält.
<code>process_ids</code>	Ein String, der die ID des hinzugefügten Prozesses enthält.

### Beispiel einer Skriptaktion

#### Beispiel

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

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

## 2.20. Skriptaktion Drucken (Print)

### Beschreibung

Das zugehörige Skript wird jedes Mal (vor) oder (nach) dem Starten eines Druckvorganges ausgeführt

Das Skript kann das Ereignis durch Zurückgeben von 0, false = VbFalse (VBScript) oder false (JScript) beenden.

### Dazugehörige Funktion

*function sa\_print(object\_id)*

Parameter	Beschreibung
<i>object_id</i>	Ein String, der die ID des Objektes enthält, auf dem der Druck zu starten ist oder gestartet wurde.

### Beispiel einer Skriptaktion

#### Beispiel

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

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

## 2.21. Skriptaktion Listendruck (PrintList)

### Beschreibung

Das zugehörige Skript wird jedes Mal vor oder nach dem Starten eines Listendrucks ausgeführt

Das Skript kann das Ereignis durch Zurückgeben von 0, false = VbFalse (VBScript) oder false (JScript) beenden.

### Dazugehörige Funktion

*function sa\_print(object\_ids)*

Parameter	Beschreibung
<i>object_ids</i>	Ein String, der die IDs der Objekte enthält, auf dem der Listendruck zu starten ist oder gestartet wurde.

### Beispiel einer Skriptaktion

#### Beispiel

```
Function sa_printlist(object_ids)
    dim x = 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
```

## 2.22. Skriptaktion OpenGraphic

### Beschreibung

Das zugehörige Skript wird jedes Mal ausgeführt, wenn ein Benutzer ein Graphikobjekt anzeigen oder bearbeiten will.

Die Objekt-ID des Objekts, die angezeigt oder bearbeitet werden soll, wird dem Skript übergeben.

Das Skript kann das Ereignis durch Zurückgeben von 0, false = VbFalse (VBScript) oder false (JScript) beenden.

### Dazugehörige Funktion

```
function sa_opengraphic(object_id)
```

### Parameter

Parameter	Beschreibung
<i>object_id</i>	Ein String, der die ID des Objektes enthält.

### Beispiel einer Skriptaktion

#### Beispiel

```
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
```



### 3. XScriptingHost Klassen

Klasse	Named Item	Zweck
<b>Klasse ScriptItemData</b>	Data	Verwaltet Daten von PPR Hub Objekten. Nimmt und setzt Attribute, durchläuft Kinderlisten, fügt Komponenten hinzu, verschiebt, kopiert und löscht Komponenten.
<b>Klasse ScriptItemQuery</b>	Query	Abfrage des PPR-Hubs.
<b>Klasse ScriptItemVersion</b>	Version	Handhabung von Objekt Versionen. Erstellen, Erproben und Gebrauch von Versionen. Setzen und Rücksetzen eines Planungsstatus
<b>Klasse ScriptItemList</b>	List	Zeigt eine Browser-Listen-Ansicht an, die als Drag & Drop Grundlage des PPR Navigators fungiert.
<b>Klasse XScriptItemDialog</b>	Dialog	Benutzerinteraktion über Dialoge – Nachrichtenfenster, Eingabefelder, Datei Auswahl
<b>Klasse ScriptItemGraphic</b>	Graphic	Empfängt die Boundingbox eines Grafikobjekts und erstellt Bitmaps (Snapshots) von 3D Objekten
<b>Klasse ScriptItemGrid</b>	Sheet	Zeigt ein Formular an
<b>Klasse SkriptItemRights</b>	Rights	Empfängt Benutzerinformation
<b>Klasse ScriptItemUnit</b>	Unit	Gibt Informationen über Einheiten zurück.
<b>Klasse ScriptItemConvert</b>	Convert	Allgemeine Umwandlungsfunktionen.
<b>Klasse ScriptItemConfig</b>	Config	Holt Information aus dem Konfigurationsmanager.
<b>Klasse ScriptItemLock</b>	CompLock	'Lock' und 'unlock' von Komponenten; gibt den gegenwärtigen Bearbeitungsstatus zurück.

### 3.1. Klasse ScriptItemData

#### Der Zugriff auf die Datenbank

Methode	Beschreibung	Verfügbarkeit
<b>AddComponent</b>	Fügt eine existierende Komponente in eine Kinderliste ein oder erzeugt eine Relation.	PE5. 13.
<b>AttributeExists</b>	Prüft, ob ein passendes Attribut (Name) für dieses Objekt existiert oder nicht.	PE5.15SP3.
<b>CacheChildIds</b>	Definiert, ob Kinder IDs gecached werden sollen oder nicht.	PE5.13.
<b>ChangeRelationship</b>	Ändert Objekte einer Relation	PE5.16.
<b>ChangeRelationshipEx</b>	Ändert Objekte einer Relation unter Berücksichtigung der erweiterten Gültigkeit	PE5.16 SP7
<b>CommitTransaction</b>	Legt die gegenwärtige Transaktion fest.	PE5.13.
<b>CopyAutoRelationUsageData</b>	Kopiert Autorelationsdaten von der Quelle zum Zielobjekt.	PE5.13.
<b>CopyComponent</b>	Kopiert eine Komponente.	PE5.13.
<b>CopyTimeAnalysis</b>	Eindeutige Kopie einer angehängten Zeitanalyse	PE5.13SP8/ PE5.14SP3.
<b>CreateComponent</b>	Erstellt eine neue Komponente (und fügt sie optional in die Kinderliste ein)	PE5.13.
<b>CreateRelationshipEx</b>	Erzeugt eine Relation zwischen Objekten, mit oder ohne Besitzer	PE5.16.
<b>CreateRelationshipWithOwner</b>	Erzeugt eine sogenannte Eigentümerbeziehung (dies ersetzt die früheren "Connectionline" Objekte).	PE5.13.
<b>CreateRootOrLibraryComponent</b>	Erzeugt eine Projekt - oder eine Bibliothekskomponente.	PE5.13.
<b>DeleteComponent</b>	Löscht eine Komponente (und entfernt sie von allen Kinderlisten). Dies ist eine Methode zum Löschen von Objekten. Mit dieser Methode müssen Sie <b>VORSICHTIG</b> umgehen, insbesondere bei Rekursionen. Der <b>delete_mode</b> bestimmt, ob nur ein Stücklisteneintrag oder auch die darunter liegende Komponente (mit-) gelöscht werden sollen.	PE5.13
<b>DisableChildrenListFilter</b>	Deaktiviert lokal alle Projektfilter, um die vollständige Liste der Kinder zu erhalten.	PE5.14
<b>ExecuteServerMethod</b>	Führt eine eindeutige Servermethode aus (Sie müssen eine Kennzeichnung übergeben).	PE5.13
<b>ExecuteServerMethodEx</b>	Führt eine eindeutige Servermethode aus (Sie müssen eine Kennzeichnung übergeben).	PE5.16SP4
<b>GetAttributebyId</b>	Mit den <b>GetAttribute/SetAttribute</b> - Methoden erhalten Sie Informationen von <b>einem</b> Objekt der Datenbank. Die Attribute erhalten Sie aus dem Konfigurationswerkzeug.	PE5.13
<b>GetAttributesbyId</b>	Wie GetAttributebyId, nur dass hier mit einem Serverzugriff Informationen (Werte) von mehreren Attributen eines Objektes erhalten werden.	PE5.13

<b>GetChildrenCount</b>	Gibt die Anzahl der Kinder in einer Liste zurück.	PE5.13
<b>GetFirstChild</b>	Empfängt die Kindliste einer Komponente und gibt das erste Kind in der Liste zurück. <b>GetFirstChild/GetNextChild</b> operieren auf einer Struktur, in welcher die Daten für verschiedene Root-Objekte und deren Stücklisten (allgemein Kindlisten) gehalten werden. GetFirstChild initialisiert die Struktur d.h. es werden alle Kinder für die übergebene Komponente und Kindliste eingelesen und liefert das erste Kind zurück. Mit GetNextChild erhalten Sie sukzessive die weiteren Kinder.	PE13
<b>GetLinkedObjectAttributebyId</b>	Holt ein einzelnes Attribut eines Objekts, das mit dieser Komponente verlinkt ist. Die Methoden <b>GetLinkedObjectAttributeById / SetLinkedObjectAttributeById</b> werden im Zusammenhang mit Relationen benötigt. Da eine Relation zwei "Väter" hat, wäre es aus der Objekt-ID der Relation alleine nicht möglich, das verknüpfte Objekt zu bestimmen. Einer der Väter der Relation ist jedoch die Wurzel der eingelesenen Kindliste. Die Objekt-ID dieses Vaters muss also zusätzlich übergeben werden, um den anderen (gesuchten) Vater zu finden. Dies ist der Parameter <b>parent_id</b> in den genannten Methoden.	PE5.13
<b>GetLinkedObjectAttributesbyId</b>	Holt sich mehrfach Attribute eines Objektes (mit einem Serverzugriff), die mit dieser Komponente verlinkt ist.	PE5.13
<b>GetNextChild</b>	Empfängt das nächste Kind aus der Kinderliste (nach dem Aufruf GetFirstChild)	PE5.13
<b>GetObjectGUIDbyObjectId</b>	Gibt die Global Unique Identifier (GUID) eines Objektes anhand seiner ID zurück.	PE5.13
<b>GetObjectIdbyObjectGUID</b>	Gibt die Objekt ID eines Objektes anhand seiner Global Unique Identifier (GUID) zurück.	PE5.13
<b>GetOtherRelatedObject</b>	Gibt das zweite „parent“ (Elternobjekt) einer Relation zurück.	PE5.13
<b>GetRelationshipsForOwnerByName</b>	Gibt die ‚Eigentümerbeziehung‘ eines eindeutigen Typen (den Relationsnamen) zurück	PE5.13
<b>IsDerivedFromClass</b>	Prüft, ob ein Objekt von einer Klasse (z. B. " XDOEr-goCompProcessDefault ") abgeleitet wird.	PE5.13
<b>IsDerivedFromType</b>	Prüft, ob ein Objekt von einem Typ (z.B. „ ergocomp-processdefault“) abgeleitet wird.	PE5.15
<b>ReadSessionData</b>	Liest die Daten (sessiondata) von HKCU/HKLM oder aus der Datenbank (Benutzerabhängig/Global)	PE5.14
<b>RemoveComponent</b>	Entfernt eine Komponente aus der Kinderliste (sollte nur in Verbindung mit einem gleichzeitigen AddComponent verwendet werden).	PE5.13
<b>ResetItem</b>	Zurücksetzen des Skriptes in seinen Anfangsstatus.	PE5.13
<b>ResetIterator</b>	Mit der Methode <b>ResetIterator</b> ist es möglich, den Index einer Kinderliste zurückzusetzen, um die Möglichkeit zu erhalten, eine Liste auch mehrfach durchlaufen zu können.	PE5.13
<b>RollbackTransaction</b>	Wiederholt die gegenwärtige Transaktion.	PE5.13
<b>SetAttributebyId</b>	Erstellt ein Attribut des Objektes	PE5.13

<b>SetAttributeRange</b>	Definiert einen Bereich für ein eindeutiges Attribut und ruft alle Objekte innerhalb dieses Bereiches ab.	PE5.14
<b>SetAttributesbyId</b>	Erstellt mehrere Attribute eines Objektes mit einem Serverzugriff	PE5.13
<b>SetFetchingSize</b>	Wählt die Anzahl der Kinder aus, die mit einem Serverzugriff geholt werden sollen.	PE5.13
<b>SetLinkedObjectAttributebyId</b>	Setzt ein einzelnes Attribut eines Objekts, das mit dieser Komponente verlinkt ist.	PE5.13
<b>SetLinkedObjectAttributesbyId</b>	Setzt mehrfach Attribute eines Objektes, das mit dieser Komponente verlinkt ist, mit einem Serverzugriff.	PE5.13
<b>SetOrderAttribute</b>	Sortieren der Kinder einer Liste nach einem vordefinierten Attribut	PE5.13
<b>WriteSessionData</b>	Schreibt die Daten (sessiondata) auf HKCU/HKLM oder auf die Datenbank (Benutzerabhängig/Global)	PE5.14

### 3.1.1. Liste der ScriptItemData Methoden

#### 3.1.1.1. AddComponent

##### Syntax

AddComponent(bstrParentObjectld, bstrChildObjectld, BstrChildListName);

Parameter	Beschreibung
bstrParentObjectld	Ein String, der die ID des Vaterobjekts oder des ersten Relationsobjektes enthält.
bstrChildObjectld	Ein String, der die ID des Kindobjekts oder das zweite Relationsobjekt enthält, das hinzugefügt werden soll.
bstrChildListName	Ein String, der den Namen der Kinderliste oder den Relationstyp enthält, der das Kind hinzugefügt werden soll.

##### Rückgabewert

Die ID des erzeugten Objekts. Die ID kann entweder von einem

- Stücklisteneintrag (= BOM; subcompitem, subcompviewitem) oder von einer
- Relation (relationship) sein.

##### Beispiel für Stücklisteneinträge (Auszug)

##### Beispiel

```
sub main(id)
...
platype = "Part"
childlistname = "nodes"
grandparent_id=Data.GetAttributebyld(id, "relationobject2")
id_1 = Data.GetFirstChild(grandparent_id, childlistname)
id_2 = Data.GetNextChild(grandparent_id, childlistname)
parent_id_1=Data.GetAttributebyld(id_1, "relationobject2")
parent_id_2=Data.GetAttributebyld(id_2, "relationobject2")
...
'Erzeugt eine Komponente in der Projektbibliothek und fügt sie
der Kinderliste "parent 1" hinzu
comp_id = Data.CreateComponent(parent_id_1, "", platype)
sci_id = Data.AddComponent(parent_id_1, comp_id, childlistname)
...
end sub
```

**3.1.1.2. AttributeExists****Syntax**

AttributeExists(*bstrSourceObjectId*, *bstrAttributeName*);

Parameter	Beschreibung
<i>bstrSourceObjectId</i>	Die ID des Quellobjektes.
<i>bstrAttributeName</i>	Ein String, der den Klassennamen des zu überprüfenden Attributes enthält. Sie können den konfigurierten sowie den tatsächlichen Klassennamen benutzen.

**Rückgabewert**

<b>-1</b>	Wenn das Attribut für dieses bestimmte Objekt existiert
<b>0</b>	Wenn das Attribut für dieses bestimmte Objekt nicht existiert.

**Beispiel (Ein Auszug)****Beispiel**

```
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
```

**3.1.1.3. CacheChildIds****Syntax**

CacheChildIds(*bFlag*);

Parameter	Beschreibung
<i>bFlag</i>	Ein globales Kennzeichen, der den Host anweist, die abgerufenen Kinder in eine Mappe zu cachen oder nicht zu cachen.

**Rückgabewert**

kein Rückgabewert.

**Beispiel (Auszug)****Beispiel**

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

### 3.1.1.4. ChangeRelationship

#### SYNTAX

**ChangeRelationship** (*lngChangeMask*, *bstrRelationshipObjId*, *bstrSourceObjId*, *bstrTargetObjId*, *bstrSourceUnexposedObjId*, *bstrTargetUnexposedObjId*, *bstrOwnerObjId*);

Parameter	Beschreibung
<i>lngChangeMask</i>	Die Bitmask spezifiziert das zu ändernde Objekt.
<i>bstrSourceObjId</i>	Die ID des Ausgangsobjektes, Quellobjekt ID.
<i>bstrTargetObjId</i>	Das Ziel der Objekt ID.
<i>bstrSourceUnexposedObjId</i>	Der Ursprung der ungeöffneten Objekt ID
<i>bstrTargetUnexposedObjId</i>	Das Ziel der ungeöffneten Objekt ID.
<i>bstrOwnerObjId</i>	Die Objekt ID vom Basisobjekt, dass den Graph besitzt.

*lngChangeMask* ist eine Bitmask mit folgenden Werten:

#### EPChangeMaskEnum

<i>cmSourceExposed</i>	=	1,
<i>cmTargetExposed</i>	=	2,
<i>cmSourceUnexposed</i>	=	4,
<i>cmTargetUnexposed</i>	=	8,
<i>cmOwner</i>	=	16

#### Rückgabewert

<i>bstrRelationshipObjId</i>	Die ID des Relationobjektes, dass erzeugt wurde.
------------------------------	--

#### Beispiel

#### Beispiel

```
REM This method creates a subassembly and two operations in the projectlibrary
REM It creates a relationship between operation1 and Subassy1
REM Afterwards it replaces the participating object operation1 of the relationship by Operation2
sub main (parent_id)
'call this script on projects only
subassy1 = CreateComp (parent_id, "", "Subassembly", "Subassy1")
operation1 = CreateComp (parent_id, "", "Operation", "Operation1")
operation2 = CreateComp (parent_id, "", "Operation", "Operation2")
' create relationship (Operation1, "") process_implements_requirement
-----> (Subassy1, "unique_id_for_req_1")
relship1 = data.CreateRelationshipEx("process_implements_requirement",
operation1, subassy1, "" -
"unique_id_for_req_1", "") -
relship1_info = data.GetAttributeById(relship1, "relationship_info")
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", "")
relship1_info = data.GetAttributeById(relship1, "relationship_info")
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
```



#### **Hinweis:**

Ändert die Objekte einer Relation. Diese Methode sollte verwendet werden, um ein oder mehrere Objekte, die in einer Relation sind, zu ändern.

### 3.1.1.5. **ChangeRelationshipEx**

#### **Syntax**

ChangeRelationshipEx (lngChangeMask, bstrRelationshipObjId, bstrSourceObjId, bstrTargetObjId, bstrSourceUnexposedObjId, bstrTargetUnexposedObjId, bstrOwnerObjId, bstrExtEffectivity);

Parameter	Beschreibung
lngChangeMask	Bitmask bestimmt das zu verändernde Objekt.
bstrSourceObjId	Die ID des Ursprungsobjekts.
bstrTargetObjId	Die ID des Zielobjekts.
bstrSourceUnexposedObjId	Die ID des nicht ausgestellten Ursprungsobjekts.
bstrTargetUnexposedObjId	Die ID des nicht ausgestellten Zielobjekts.
bstrOwnerObjId	Die Objekt ID vom Grundobjekt, dass ein Diagramm besitzt.
bstrExtEffectivity	Die gegebene erweiterte Gültigkeit.

lngChangeMask ist eine Bitmask mit folgenden Werten:

#### **EPChangeMaskEnum**

cmSourceExposed	=	1,
cmTargetExposed	=	2,
cmSourceUnexposed	=	4,
cmTargetUnexposed	=	8,
cmOwner	=	16

#### **Rückgabewert**

bstrRelationshipObjId = Die ID des Relationobjektes, dass erzeugt wurde..

#### **Beispiel**

#### **Beispiel**

```
REM This method creates a subassembly and two operations in the
projectlibrary
REM It creates a relationship between Operation1 and Subassy1
REM Afterwards it replaces the participating object Operation1
of the relationship by Operation2.
sub main (parent_id)
  ' call this script on projects only
  subassy1 = CreateComp (parent_id, "", "Subassembly", "Subassy1")
  operation1 = CreateComp (parent_id, "", "Operation", "Operation1")
  operation2 = CreateComp (parent_id, "", "Operation", "Operation2")
  ' create relationship (Operation1, "") ----- proc-
ess_implements_requirement -----> (Subassy1,
"unique_id_for_req_1")
  relship1 = data.CreateRelationshipEx("process_implements_requirement",
operation1, subassy1, ""
"unique_id_for_req_1", "")
```



```

relship1_info = data.GetAttributeByld(relship1, "relationshipinfo")
MsgBox "relationship successfully created: " & vbNewLine & rel-
ship1_info
' change given relationship to
' create relationship (Operation2, "") ----- proc-
ess_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.GetAttributeByld(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.SetAttributebyld comp, "name", namecomp
CreateComp = comp
end function

```



**Hinweis:** Ändert die Objekte einer Relation. Diese Methode sollte benutzt werden um ein oder mehrere teilnehmende Objekte einer Relation zu ändern. Die Methode führt auch Konsistenzprüfungen durch, die nicht gemacht werden, wenn teilnehmende Objekte mit der Methode SetAttribute(s) geändert werden.

### 3.1.1.6. CommitTransaction

#### Syntax

CommitTransaction(**bstrObjectId**);

Parameter	Beschreibung
<b>bstrObjectId</b>	Ein String, der jede Objekt-ID innerhalb der derzeitigen Transaktion enthält.

#### Rückgabewert

kein Rückgabewert

#### Beispiel

#### Beispiel

```

REM Beispiel für commit und rollback einer Transaktion
sub main (id)

object_id = Data.GetAttributebyld (id, "relationobject2")

call Data.CommitTransaction(object_id)

' Change an attribute, make a rollback,
' Change the attribute again and commit the changes
value = Data.GetAttributebyld(object_id, "name")
value_false = value+"_JS"
call Data.SetAttributebyld(object_id, "name", value_false)
call Data.RollbackTransaction(object_id)
value_true = value+"_VBS"
call Data.SetAttributebyld(object_id, "name", value_true)
call Data.CommitTransaction(object_id)
end sub

```



**Hinweis:** Wird der Funktionsanruf innerhalb einer Skriptaktion gestartet, kann es zu konkurrierenden CommitTransaction/RollbackTransaction Aufrufen des Clients kommen. In solchen Fällen können korrupte Datenbankobjekte entstehen. Bitte rufen Sie diese Funktion deshalb nicht innerhalb einer Skriptaktion auf.

**3.1.1.7. CopyAutoRelationUsageData****SYNTAX**

CopyAutoRelationUsageData(*bstrSourceObjectId*,  
*bstrTargetObjectId*);

Parameter	Beschreibung
<i>bstrSourceObjectId</i>	Die ID des <i>Quell</i> objektes.
<i>bstrTargetObjectId</i>	Die ID des <i>Ziel</i> objektes.

**Rückgabewert**

Keiner

**Beispiel:****Beispiel**

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

**3.1.1.8. CopyComponent****Syntax**

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

Parameter	Beschreibung
<i>BstrParentObjectId</i>	Ein String, der die ID des Vaterobjekts enthält.
<i>bstrChildCopyObjectId</i>	Ein String, der die ID des Kindobjekts enthält, das kopiert werden soll.
<i>iCopyMode</i>	Ein Integer, das den Kopiermechanismus definiert.

**Mögliche Werte von *iCopyMode***

0	Copy normal	Kopiert nur die Komponente.
1	Copy flat	Kopiert die Komponente und ihre Kinder (flach).
2	Copy deep	Kopiert die Komponente und ihre Kinder (tief).

**Rückgabewert**

Die ID des erzeugten Objektes.

**Beispiel (Auszug)****Beispiel**

```
sub main(i d)
...
pl antype = "Part"
ch ildl istname = "nodes"
grandparent_id=Data.GetAttributebyId(i d, "relationobject2")
i d_1 = Data.GetFirstChild(grandparent_id, ch ildl istname)
i d_2 = Data.GetNextChild(grandparent_id, ch ildl istname)
parent_id_1=Data.GetAttributebyId(i d_1, "relationobject2")
parent_id_2=Data.GetAttributebyId(i d_2, "relationobject2")
...
' Make a copy of the last created component at parent 2
sci_id = Data.CopyComponent(parent_id_2, comp_id, 0)
```

```
....  
end sub
```

### 3.1.1.9. CopyTimeAnalysis

#### Syntax

CopyTimeAnalysis(*bstrSourceObjectId*, *bstrTargetObjectId*);

Parameter	Beschreibung
<i>bstrSourceObjectId</i>	Die ID des <i>Quell</i> objektes.
<i>bstrTargetObjectId</i>	Die ID des <i>Ziel</i> objektes.

#### Rückgabewert

Keiner

#### Beispiel (ein Auszug)

### Beispiel

```
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,  
"ergcompbase")  
call Data.CopyTimeAnalysis(source_process_id, target_process_id)<  
....
```



---

**Hinweis:** Diese Funktion ist nur anwendbar, wenn Zeitanalysen mittels Finder-Control mit einem Prozess verknüpft sind. Sie ist nicht in der Integrierten Zeitwirtschaft anwendbar (integrated standard time measurement client).

---

**3.1.1.10. CreateComponent****Syntax**

CreateComponent(bstrParentObjectId, bstrChildListName, bstrChildName);

Parameter	Beschreibung
bstrParentObjectId	Ein String, der die ID des Vaterobjekts enthält.
bstrChildListName	Ein String, der den Namen der Kinderliste enthält, der das Kind hinzugefügt werden soll. (Normalerweise „nodes“). Wenn der Parameter "" (ein leerer String) ist, wird das Objekt in der Projektbibliothek erstellt.
bstrChildName	Ein String, der den Namen des Typs oder Plantyps enthält, dem das Kind hinzugefügt werden soll.

**Rückgabewert**

Die ID des erzeugten Objekts.

**Beispiel****Beispiel**

```
sub main(id)
...
pl antype = "Part"
childlistname = "nodes"
grandparent_id=Data.GetAttributebyId(id, "relationobject2")
id_1 = Data.GetFirstChild(grandparent_id, childlistname)
id_2 = Data.GetNextChild(grandparent_id, childlistname)
parent_id_1=Data.GetAttributebyId(id_1, "relationobject2")
parent_id_2=Data.GetAttributebyId(id_2, "relationobject2")
...
'Erzeugt Komponenten als Kinder von parent 1
sci_id = Data.CreateComponent(parent_id_1, childlistname, pl an
type)
...
end sub
```

**Hinweis:**

Nur für Kinder, die mit "nodes" erzeugt werden.

*Bevor diese Methode ausgeführt wird, muss geprüft werden, ob der zu erstellende Planungstyp (oder Typ) existiert und ob es Eltern-Kind-Relation dazu gibt.*

**3.1.1.11. CreateRelationshipEx****SYNTAX**

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

Parameter	Beschreibung
bstrRelationName	Der Name der zu erzeugenden Relation (z. B. "process_runsbefore_process")
bstrSourceObjectId	Die ID des Ausgangsobjektes, Quellobjekt ID.
bstrTargetObjectId	Das Objekt ID des Ziels.
bstrSourceUnexposedObjectId	Der Ursprung der ungeöffneten Objekt ID
bstrTargetUnexposedObjectId	Das Ziel der ungeöffneten Objekt ID.
bstrOwnerObjectId	Die Objekt ID vom Basisobjekt, die den Graph besitzt.

**Beispiel****Beispiel**

```

REM This method creates a subassembly and two operations in the
projectlibrary
REM It creates a relationship between operation1 and Subassy1
REM Afterwards it replaces the participating object operation1
of the relationship by operation2
sub main (parent_id)
' call this script on projects only
subassy1 = CreateComp (parent_id, "", "Subassembly", "Subassy1")
operation1 = CreateComp (parent_id, "", "Operation", "Operation1")
operation2 = CreateComp (parent_id, "", "Operation", "Operation2")
' create relationship (Operation1, "") process_implements_requirement -
-> (Subassy1, "unique_id_for_req_1")
relship1 =
data.CreateRelationshipEx("process_implements_requirement", op-
eration1, subassy1, "", "unique_id_for_req_1", "")
relship1_info = data.GetAttributeById(relship1, "relationship-
info")
MsgBox "relationship successfully created: " & vbNewLine & rel-
ship1_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", "")
relship1_info = data.GetAttributeById(relship1, "relationship-
info")
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

```

**Rückgabewert**

bstrRelationshipObjectId	Die ID der Relation des Objektes, dass erzeugt wurde.
--------------------------	---

**3.1.1.12. CreateRelationshipWithOwner****SYNTAX**

```
CreateRelationshipWithOwner(bstrRelationName,
                             bstrSourceObjectId,
                             bstrTargetObjectId,
                             bstrOwnerObjectId);
```

Parameter	Beschreibung
RelationName	Der Name der zu erzeugenden Relation (z. B. "process_runsbefore_process").
SourceObjectId	Die ID des Quellobjektes.
TargetObjectId	Die ID des Zielobjektes.
OwnerObjectId	Die Objekt-ID des basis Objektes (im Graphen).

**Rückgabewert**

bstrRelationshipObjectId	Die ID des erzeugten Relationsobjektes.
--------------------------	---

**Beispiel für CreateRelationshipWithOwner****Beispiel**

```
Dim parent_process_pt
Dim child_process_pt
Dim relationship_name
' *****
sub main(id)
    parent_process_pt = "Workplan"
    child_process_pt = "Operation"
    relationship_name = "process_runsbefore_process"

    parent_proc_id = GetBase(id)
    child1_proc_sci_id = CreateSubProcess(parent_proc_id)
    child2_proc_sci_id = CreateSubProcess(parent_proc_id)

    child1_proc_id = GetBase(child1_proc_sci_id)
    child2_proc_id = GetBase(child2_proc_sci_id)

    pg_id = CreatePG(parent_proc_id)

    scvip1_id = CreateSCVIP(child1_proc_sci_id)
    scvip2_id = CreateSCVIP(child2_proc_sci_id)

    call Data.SetAttributebyId(scvip1_id, "posx", 3)
    call Data.SetAttributebyId(scvip1_id, "posy", 5)

    call Data.SetAttributebyId(scvip2_id, "posx", 5)
    call Data.SetAttributebyId(scvip2_id, "posy", 5)

    relationship_id = Data.CreateRelationshipWithOwner(relationship_name,
    child1_proc_id, child2_proc_id, parent_proc_id)
    call SetConnectonLineOrigin(relationship_id, cint(1), cint(2))
end sub
' *****
function CreateSubProcess(process_id)
    child_id = Data.CreateComponent(process_id, child_process_pt,
    child_process_pt)
    CreateSubProcess = child_id
end function
' *****
function CreatePG(process_id)
    CreatePG = Data.CreateComponent(process_id, "processgraph", "process-
    graph")
end function
' *****
function GetBase(id)
    GetBase = Data.GetAttributebyId(id, "relationobject2")
end function
' *****
function CreateSCVIP(sci_id)
```

```

' check for already existing
call Query.ResetSearch
call Query.SetQuery("subcompvi ewi temi tempro", "m_pSubCompl tem", "=",
sci_id)
scviip_id = Query.GetFirstResult
if scviip_id = "" then
parent_ecb_id = Data.GetAttributebyId(sci_id, "relationobject1")
graph_bom_id = Data.GetAttributebyId(parent_ecb_id, "graph bom")
scviip_id = Data.AddComponent(graph_bom_id, sci_id, "subcompvie-
wi temi tempro")
end if
CreateSCVIIP = scviip_id
end function
' *****
sub SetConnectionLineOrigin(rel_id, i Begin, i End)
call Data.SetAttributebyId(rel_id, "connectionpointsource", i Begin)
call Data.SetAttributebyId(rel_id, "connectionpointsource", i End)
end sub

```




---

**Hinweis:** Mit dieser Funktion können Relationen im Prozessgraph erzeugt werden. Diese Funktion ersetzt ab der Version PE 5.10 die Funktion "connectionlines".

---

### 3.1.1.13. CreateRootOrLibraryComponent

#### SYNTAX

CreateRootOrLibraryComponent(*bstrRootObject*,  
*bstrParam*);

Parameter	Beschreibung
<i>bstrRootObject</i>	Die Kennzeichnung für das Wurzelobjekt. Mögliche Werte sind "ProjectRoot" oder "ArchivRoot".
<i>bstrParam</i>	Wenn Sie ein Projekt erzeugen wollen, also wenn <i>bstrRootObject</i> = "ProjectRoot" ist, müssen Sie die ID des Master-Planungstypensatzes (Planungstypensatz in der Systembibliothek) angeben.

#### Rückgabewert

Die ID des erzeugten Objektes. Dies ist entweder die ID einer

- Projekt- oder
- Bibliothekskomponente

#### Beispiel:

#### Beispiel

```
...
function CreateProject
Dim PTSCombo
Query.ResetSearch
Call Query.SetQuery("ergopl antypeset", "m_plMaster", "=", Empty)
Call Query.SetConcatenator("AND")
Call Query.SetQuery("ergopl antypeset", "tablename", "<>", "ergopl antype-
set_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", "Pl antypesets", _
PTSCombo)
ret = Dialog.InputBox("Select a pl antypeset")
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", mas-
ter_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
End If
```



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



---

**Achtung:** Mit dieser Funktion können Sie neue Projekte und Komponenten in der Systembibliothek (die dann unabhängig von Projekten sind) erzeugen - einige Objekte müssen aber richtig initialisiert werden, um auf der Benutzeroberfläche sichtbar zu sein. Verwenden Sie deshalb die Funktion CreateRootOrLibraryComponent sehr vorsichtig. Falls Sie Bedenken bei der Handhabung der Funktion haben, setzen Sie sich bitte mit unserer Anwender-Hotline in Verbindung: Telefon: +49 / 711 / 27 300-400;

Telefax: +49 / 711 / 27 300-599 oder

E-Mail: [support@delmia.de](mailto:support@delmia.de)

---

**3.1.1.14. DeleteComponent****Syntax**

DeleteComponent(*bstrParentObjectId*, *iDeleteMode*);

Parameter	Beschreibung
<i>bstrParentObjectId</i>	Ein String, der die ID des Vaterobjekts enthält.
<i>iDeleteMode</i>	Ein Integer, der den Typ des Löschmechanismus definiert. (siehe Hinweis)

**Rückgabewert**

kein Rückgabewert

**Beispiel (Auszug)****Beispiel**

```

sub main(i d)
...
retval = Dialog.MessageBoxExt("Option for delete all created components", "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
    Else
        Delete_id = sci_ids(i)
        End if
        call Data.DeleteComponent(sci_ids(i), delete_flag)
    next
end if
...
end sub

```

**Achtung:**

Vorsicht ist geboten wenn diese Funktion in Rekursionen oder Schleifen benutzt wird und vor allen Dingen wenn Tief(1 (Deep) gelöscht werden soll.

iDeleteMode	Object base configuration type	Result
0 (Flat)	subcompitem	
(Flat) or 1 (Deep)	relationship	Löscht relationships alleine.
0 (Flat) or 1 (Deep)	ergoitem/dodefaultimpl	Löscht ergoitems aus der Projektbibliothek.
0 (Flat)	ergocompbase	Removes ergocompbase from project library and all referencing subcompitem.
1 (Deep)	ergocompbase oder subcompitem	Löscht ergocompbase Objekte und deren Kinder aus der Projektbibliothek und alle referenzierten subcompitem.

### 3.1.1.15. DisableChildrenListFilter

#### Syntax

SetOrderAttribute(*bstrAttribute*, *bFlag*);

Parameter	Beschreibung
<i>bstrAttribute</i>	Das Attribut, für welches der Bereich definiert ist.
<i>bFlag</i>	<i>True</i> , wenn die Kinderliste ungefiltert zurückgegeben werden soll, ansonsten <i>False</i> .

#### Rückgabewert

Keiner

#### Beispiel

#### Beispiel

```
sub main(id)
    parent_base_id = Data.GetAttributeById(id, "relationobject2")
    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, "relationobject2")
            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
```

Sie haben unterschiedliche Möglichkeiten Kinderlisten zu erhalten. Diese Möglichkeiten können sich gegenseitig ausschließen:

- Standard (Entsprechend den Filtereinstellungen im Projekt, nicht sortiert)
- Sortiert (Entsprechend den Filtereinstellungen im Projekt, sortiert), siehe auch die Funktion SetOrderAttribute
- **Ungefiltert (nicht sortiert).**

- Bereich (ungefiltert, unsortiert, nur Objekte deren Attributwert innerhalb eines vordefinierten Bereichs liegen, werden zurückgegeben). Siehe auch [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.

### 3.1.1.16. *ExecuteServerMethod*

#### SYNTAX

ExecuteServerMethod ([bstrObjectId](#), [bstrMethod](#));

Parameter	Beschreibung
<a href="#">bstrObjectId</a>	Die Objekt ID, für die die Funktion ausgeführt werden soll.
<a href="#">bstrMethod</a>	Eine Zeichenfolgekennzeichnung, die die Funktion beschreibt, die ausgeführt werden soll.

#### Beispiel

#### Beispiel

```
sub main(i d)
  call Data.ExecuteServerMethod(i d, "freqcalc")
end sub
```



**Hinweis:** Dieser Aufruf löst eine spezielle Implementierung im Server aus, der auf den Objekttyp des übergebenen Objektes (ID) beschränkt sein könnte. Die Kennzeichnung der Funktion (z. B. "Freqcalc") gibt die Bearbeitungsstelle an.

### 3.1.1.17. ExecuteServerMethodEx

#### SYNTAX

ExecuteServerMethodEx (bstrObjectId, bstrMethod, pVarArrayIn);

Parameter	Beschreibung
bstrObjectId	Die Objekt ID, für die die Funktion ausgeführt werden soll.
bstrMethod	Ein Feld, die die Funktion beschreibt, die ausgeführt werden soll.
pVarArrayIn	Ein Feld, welches die Parameter enthält.

#### Beispiel Beispiel

```
sub main(i d)
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) = i d

outArray = data.ExecuteServerMethodEx (i d,
"key_copy_return_objects", inArray)

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



**Hinweis:** Dieser Aufruf löst eine spezielle Implementierung im Server aus, der auf den Objekttyp des übergebenen Objektes (ID) beschränkt sein könnte. Die Kennzeichnung der Funktion (z. B. "Freqcalc") gibt die Bearbeitungsstelle an.

**3.1.1.18. GetAttributebyId****Syntax**

GetAttributebyId(*bstrObjectId*, *bstrName*);

Parameter	Beschreibung
<i>bstrObjectId</i>	Eine Zeichenfolge (String), die die Objekt-ID enthält.
<i>bstrName</i>	Ein String, mit den Namen des Attributs, das erhalten werden soll.

**Rückgabewert**

Wert des Attributs.

**Beispiel****Beispiel**

```
REM Zeigt den aktuellen Name der Komponente an und
REM hängt den String "_VB" an.
Sub main (id)
    object_id=Data.GetAttributebyId(id, "relationobject2")
    value = Data.GetAttributebyId(object_id, "name")
    value = value+"_VB"
    Call Data.SetAttributebyId(object_id, "name", value)
End sub
```

**Hinweis:** Bitte beachten Sie, dass get/set Attribute(s) von Subcompitem(s) und Relationship Objekten meistens zu einer Ergocompbase durch eine Anwendung des Servers umgeleitet werden.

**3.1.1.19. GetAttributesbyId****Syntax**

GetAttributesbyId(*bstrObjectId*, *pvarArrayOfNames*);

Parameter	Beschreibung
<i>bstrObjectId</i>	Ein String, der Die ID des Objekts enthält.
<i>pvarArrayOfNames</i>	Ein Feld, das die Namen der Attribute enthält, die empfangen werden sollen.

**Beispiel****Beispiel**

```
REM Erhält die Attribute über die ID, mit einen Datenbankzugriff.
REM Attribute: 1-dim Feld von Attributen Namen
REM Werte: 2-dim Feld von Ergebnissen, dimy ist 1 (attributen/Werte
REM alle Felder basieren auf Null, z.B. attributes(3) bedeutet 4 Elemente
REM sind im Feld.
REM Hinweis: Die Werte im Feld sind nach Attributen sortiert!
```

```
Sub main(id)
    dim attributes (3)
    attributes(0)="name"
    attributes(1)="nameshort"
    attributes(2)="creationdate"
    attributes(3)="modificationdate"

    values = Data.GetAttributesbyId(id, attributes)
    dim x = ubound(values, 1)
    dim y = ubound(values, 2) ' must be 1 (!)
    For i = 0 to dim x
        attribute = values(i, 0)
        value = values(i, 1)
        MsgBox(attribute & ": " & value)
    Next
End sub
```

**3.1.1.20. GetLinkedObjectAttributebyId****Syntax**

GetLinkedObjectAttributebyId( bstrLinkId, bstrParentObjectId, bstrName);

Parameter	Beschreibung
bstrLinkId	Ein String, der die ID des Relationship Objektes enthält.
bstrParentObjectId	Ein String, der die ID eines Vaters des Relationship Objektes enthält.
bstrName	Ein String, der den Namen des zu empfangenden Attributs enthält.

**Rückgabewert**

Der derzeitige Wert des Attributes.

**Beispiel****Beispiel**

```
REM Dieses Skript veranschaulicht die Benutzung von
REM Get/SetLinkedObjectAttribute(s)byId
Sub main(parent_id)
    relationname="plant_provides_prod"
    parent_base_id=Data.GetAttributebyId(parent_id, "relationship2")

    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
```



**Hinweis:** Benutzen Sie diese Methode, wenn sie ein Objekt vom Typ Relationship und einen seiner Väter kennen, und ein Attribut des anderen Vaters erhalten möchten.

**3.1.1.21. GetLinkedObjectAttributesbyId****Syntax**

```
GetLinkedObjectAttributesbyId(bstrLinkId,  
                               bstrParentObjectId,  
                               pvarArrayOfNames,  
                               pVarArrayOfValues);
```

Parameter	Beschreibung
<b>bstrLinkId</b>	Ein String, der die ID des Relationship-Objekts enthält.
<b>bstrParentObjectId</b>	Ein String, der die ID eines Vaters der Relationship-Objekte enthält.
<b>pvarArrayOfNames</b>	Ein Feld, das die Namen der zu empfangenden Attribute enthält.
<b>pVarArrayOfValues</b>	Ein Feld, das die empfangenen Attribute / Wertepaare enthält.

**Rückgabewert**

Ein Feld (array) für die erhaltenen Attribute / Wertepaare.

**Beispiel**

```
REM Ein Skript, das den Gebrauch von Get/SetLinked Objekt At-  
tributen über ID, aufzeigt.  
REM Empfängt die Namen von Objekten (Typ A), die über Relati-  
onsnamen auf Objekte (Typ B) verweisen.  
REM Um den Code nicht zu komplex werden zu lassen, wird nur ein  
Kind abgerufen  
sub main(parent_id)  
    relationname="plant_provides_prod"  
    parent_base_id=Data.GetAttributebyId(parent_id, "relationobject2")  
    child_id = Data.GetFirstChild(parent_base_id, relationname)  
    if child_id <> "" Then  
        REM Mehrfach Attribute  
        dim attributes (3)  
        attributes(0)="name"  
        attributes(1)="nameshort"  
        attributes(2)="creationdate"  
        attributes(3)="modificationdate"  
        values = Data.GetLinkedObjectAttributesbyId(child_id, par-  
ent_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
```



**3.1.1.22. GetFirstChild****Syntax**

```
GetFirstChild(bstrParentObjectId,  
             bstrChildListName);
```

Parameter	Beschreibung
<b>bstrParentObjectId</b>	Ein String, der die Objekt ID des Vaters enthält.
<b>bstrChildListName</b>	Ein String, der den Name der Kinderlisten enthält, von der die Kinder geholt werden sollen.

**Rückgabewert**

Die Objekt-ID des ersten Kindes der Kinderliste oder ein Leerstring, wenn keine Kinder in der Liste vorhanden sind.

**Beispiel****Beispiel**

```
REM Ein Skript, das den Gebrauch von Get/SetLinked Objekt über  
die ID aufzeigt.  
REM Empfängt die Namen von Objekten (Typ A), die über Relati-  
onsnamen auf Objekte (Typ B) verweisen.  
REM Um den Code einfach zu halten, wird nur ein Kind abgerufen  
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("Anzahl der Kinder: " & 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
```

**3.1.1.23. GetNextChild****Syntax**

```
GetNextChild( bstrParentObjectId,
              bstrChildListName);
```

Parameter	Beschreibung
<code>bstrParentObjectId</code>	Ein String, der die Objekt ID des Vaters enthält.
<code>bstrChildListName</code>	Ein String, der den Kinderlistenamen enthält, von der die Kinder geholt werden sollen.

**Rückgabewert**

Die Objekt ID des ersten Kindes der Kinderliste oder ein Leerstring, wenn keine Kinder in der Liste vorhanden sind.

**Beispiel****Beispiel**

```
REM Veranschaulicht das Holen der Kinder aus der Kinderliste
"nodes"
REM a) Die Kinder werden nach dem Namen sortiert(aufsteigend)
aufgelistet
REM b) Vor dem Abrufen der Kinder wird die Anzahl der Kinder
angezeigt
REM c) alle Kinder werden in der "Do While Schleife" abgeholt.
REM d) nach dem Abholen aller Kinder wird der Iterator wieder
auf den Anfang der Liste gesetzt.
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
```

### 3.1.1.24. GetChildrenCount

#### Syntax

GetChildrenCount (bstrParentObjectId,  
bstrChildListName);

Parameter	Beschreibung
bstrParentObjectId	Ein String, der die Objekt ID des Vaters enthält.
bstrChildListName	Ein String, der den Namen der Kinderliste enthält, von der die Kinder geholt werden sollen.

#### Rückgabewert

Die Anzahl der Kinder der Kinderliste oder Null bei einer leeren Kinderliste.

#### Beispiel

### Beispiel

```
REM Veranschaulicht das Holen der Kinder aus der Kinderliste
"nodes"
REM a) Die Kinder werden nach Namen sortiert(aufsteigend) aufgelistet
REM b) Vor dem Abrufen der Kinder wird die Anzahl der Kinder angezeigt
REM c) alle Kinder werden in der "Do While Schleife" abgeholt.
REM d) nach dem Abholen aller Kinder wird der Iterator wieder auf den Anfang der Liste gesetzt.
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
```

**3.1.1.25. GetObjectGUIDbyObjectId****SYNTAX**

GetObj ectGUI DbyObj ectI d(*bstrObj ectI d*);

Parameter	Beschreibung
<i>bstrObjectId</i>	Die Objekt ID.

**Rückgabewert**

<i>bstrObjectGUID</i>	Die einzigartige, globale Kennzeichnung ( <i>GUID= Global Unique Identifier</i> ).
-----------------------	--

**Beispiel****Beispiel**

```
sub mai n(project_i d)
  project_gui d = Data. GetObj ectGUI DbyObj ectI d(project_i d)
  MsgBox("The projects GUID is: " & project_gui d)
  project_i d2 = Data. GetObj ectI dbyObj ectGUI D(project_gui d,
"ergoproject")
  if project_i d = project_i d2 then
    MsgBox("Success. The returned ID is identical to the
entry id.")
  else
    MsgBox("Failure. ")
  end if
end sub
```



**Hinweis:** Um ein Objekt zu identifizieren, verwendet die Software zwei verschiedene ID's, die (POET) Objekt ID und eine generierte globale einzigartige Kennzeichnung (GUID= Global Unique Identifier). Beim Upgrade oder dem Wechseln der Datenbank kann die (POET) Objektkennung einer Änderung unterzogen werden, während die GUID immer konstant bleibt.

**3.1.1.26. GetObjectldbyObjectGUID****SYNTAX**

GetObj ectI dbyObj ectGUI D(*bstrObj ectGUI D*, *bstrTypeName*);

Parameter	Beschreibung
<i>bstrObjectGUID</i>	Die einzigartige, globale Kennzeichnung ( <i>GUID= Global Unique Identifier</i> ).
<i>bstrTypeName</i>	Der Typenname des Objektes.

**Rückgabewert**

<i>bstrObjectld</i>	Die Objekt-ID.
---------------------	----------------

**Beispiel****Beispiel**

```
sub mai n(proj ect_i d)
  project_gui d = Data. GetObj ectGUI DbyObj ectI d(proj ect_i d)
  MsgBox("The projects GUID is: " & project_gui d)
  project_i d2 = Data. GetObj ectI dbyObj ectGUI D(proj ect_gui d, "ergoproj ect")
  if project_i d = project_i d2 then
    MsgBox("Success. The returned id is identical to the entry
    i d. ")
  El se
    MsgBox("Fai lure. ")
  end if
end sub
```



**Hinweis:** Um ein Objekt zu identifizieren, verwendet die Software zwei verschiedene ID's, die (POET) Objekt ID und eine generierte globale einzigartige Kennzeichnung (GUID= Global Unique Identifier). Beim Upgrade oder dem Wechseln der Datenbank kann die (POET) Objektkennung einer Änderung unterzogen werden, während die GUID immer konstant bleibt.

**3.1.1.27. GetOtherRelatedObject****SYNTAX**

GetOtherRelatedObject(*bstrRelationshipObjId*,  
*bstrObjectOneld*);

Parameter	Beschreibung
<i>bstrRelationshipObjId</i>	Die Objekt ID des <i>relationship</i> Objektes.
<i>bstrObjectOneld</i>	Die Objekt ID des "ersten" Objektes.

**Rückgabewert**

<i>bstrObjectTwold</i>	Die Objekt ID des "zweiten" Objektes ("the other object").
------------------------	--

**Beispiel****Beispiel**

```
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, "relationobject2")
end function

' *****
function GetName(id)
    GetName = Data.GetAttributebyId(id, "name")
end function
```



**Hinweis:** Diese Funktion erleichtert nur den Zugriff auf die ID des „zweiten“ Objektes (das „erste“ Objekt ist bekannt). Mit *GetLinkedObjectAttributebyId* können Sie auch auf dieses Objekt zugreifen.

**Beispiel:**

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

**3.1.1.28. GetRelationshipsForOwnerByName****SYNTAX**

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

Parameter	Beschreibung
<i>bstrSourceObjectId</i>	Die ID des Kind-Objektes, von welchem die Relationen zu anderen Kindern geholt werden sollen.
<i>bstrOwnerObjectId</i>	Die Basis-ObjektID der Komponente, von der der Graph erzeugt wurde. (im Beispiel <code>parent_process_id</code> ).
<i>bstrRelationName</i>	Der Name der Relation z. B.. "process_runsbefore_process".

**Rückgabewert**

<i>vRelationShipIds</i>	Ein Feld von Objekt-IDs, welches alle Beziehungen zwischen einem bestimmten Kind und den anderen Kindern des Graphen beinhaltet.
-------------------------	--

**Beispiel****Beispiel**

```
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, par-
        ent_process_id, relationship_name)

        dim x = ubound(rel_ids)
        if dim x >= 0 then
            for i=0 to dim x
                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
```



**Hinweis:** Über diese spezielle Funktion hinaus können weitere Eigenschaften von Relationsobjekten mittels einer Query zurückgeholt werden.

### 3.1.1.29. IsDerivedFromClass

#### Syntax

IsDerivedFromClass(*bstrObjectId*, *bstrClassName*);

Parameter	Beschreibung
<i>bstrObjectId</i>	Ein String, der die ID des Objekts enthält.
<i>bstrName</i>	Ein String, der den zu prüfenden Klassennamen enthält (es kann sowohl der konfigurierte Name als auch der reale Klassenname übergeben werden).

#### Rückgabewert

-1	Wenn das Objekt von dieser Klasse abgeleitet ist
0	Wenn das Objekt nicht von dieser Klasse abgeleitet ist.



**Hinweis:** In VBScript wird -1 zu True oder vbTrue ausgewertet, in JScript wird +1 zu true ausgewertet.

#### Beispiel

#### Beispiel

```
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 = "Das Objekt <" + id + "> (" + name + ") " + flag
    MsgBox(info)
end sub
```



### 3.1.1.30. *IsDerivedFromType*

#### Syntax

**IsDerivedFromType**(**BstrObjectId**,**bstrTypeString**)

Parameter	Beschreibung
<b>bstrObjectId</b>	Ein String, der die ID des Objekts enthält.
<b>bstrTypeString</b>	Ein String, der den zu prüfenden konfigurierten Typennamen enthält.

#### Beispiel

#### Beispiel

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

### 3.1.1.31. ResetIterator

#### Syntax

ResetIterator (bstrParentObjectId, bstrChildListName);

Parameter	Beschreibung
bstrParentObjectId	Ein String, der die Objekt ID des Vaters enthält.
bstrChildListName	Ein String, der den Name der Kinderliste enthält, von der die Kinder geholt werden sollen.

#### Rückgabewert

Der Iterator der Kinderliste - der bei der Vaterobjekt ID und dem Namen der Kinderliste definiert ist - wird nach dem Abholen aller Kinder wieder auf den Anfang der Kinderliste gesetzt.

#### Beispiel

#### Beispiel

```
REM Veranschaulicht das Holen der Kinder aus der Kinderliste "nodes"
REM a) Die Kinder werden nach Namen sortiert(aufsteigend) aufgelistet
REM b) Vor dem Abrufen der Kinder wird die Anzahl der Kinder angezeigt
REM c) alle Kinder werden in einer "Do - While Schleife" abgeholt.
REM d) nach dem Abholen aller Kinder wird der Iterator wieder
auf den Anfang der Liste gesetzt.
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
```

**3.1.1.32. RemoveComponent****Syntax**

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

Parameter	Beschreibung
<i>bstrParentObjectId</i>	Ein String, der die ID des Vaterobjekts enthält.
<i>bstrChildObjectId</i>	Ein String, der die ID des zu löschenden Kindobjekts enthält.
<i>bstrChildListName</i>	Ein String, der den Namen der Kinderliste enthält, aus der das Kind gelöscht werden soll.

**Rückgabewert**

kein Rückgabewert

**Beispiel (Auszug)****Beispiel**

```
sub main(id)
...
platype = "Part"
childlistname = "nodes"
grandparent_id=Data.GetAttributebyId(id, "relationobject2")
id_1 = Data.GetFirstChild(grandparent_id, childlistname)
id_2 = Data.GetNextChild(grandparent_id, childlistname)
parent_id_1=Data.GetAttributebyId(id_1, "relationobject2")
parent_id_2=Data.GetAttributebyId(id_2, "relationobject2")
...

'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
```

**WARNING:** 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](#)

### 3.1.1.33. RollbackTransaction

#### Syntax

RollbackTransaction(**bstrobjectid**);

Parameter	Beschreibung
<b>bstrobjectid</b>	Ein String, der jede Objekt- ID innerhalb der derzeitigen Transaktion enthält.

#### Rückgabewert

kein Rückgabewert

#### Beispiel

#### Beispiel

```
REM Beispiel für commit and rollback einer Transaktion
sub main (id)

    object_id=Data.GetAttributebyid(id, "relationobject2")

    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
```



**Hinweis:** Wird der Funktionsanruf innerhalb einer Skriptaktion gestartet, kann es zu konkurrierenden CommitTransaction / RollbackTransaction Aufrufen des Clients kommen. In solchen Fällen können korrupte Datenbankobjekte entstehen. Bitte rufen Sie diese Funktion deshalb nicht innerhalb einer Skriptaktion auf.

**3.1.1.34. SetOrderAttribute****Syntax**

SetOrderAttribute(**bstrAttribute**name, **bstrChildList**name);

Parameter	Beschreibung
<b>bstrAttribute</b> name	Ein Attribut, das der Sortierschlüssel ist oder ein leerer String, wenn eine Sortierung nicht benötigt wird.
<b>BstrChildList</b> name	Entweder "ASC" (aufsteigend) or "DESC" (absteigend).

**Rückgabewert**

kein Rückgabewert

**Beispiel****Beispiel**

```

Sub main(parent_id)
    parent_base_id = Data.GetAttributeById(id, "relationobject2")
    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, default_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, "relationobject2")
            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

```

**3.1.1.35. SetAttributebyId****Syntax**

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

Parameter	Beschreibung
<i>BstrObjectId</i>	Ein String, der die ID des Objekts enthält.
<i>BstrName</i>	Ein String, der den Namen des Attributs enthält, das gesetzt werden soll.
<i>newValue</i>	Eine Variable des Typs Variant, der den neuen Wert des Attributs enthält.

**Rückgabewert**

Kein Rückgabewert

**Beispiel****Beispiel**

```
REM Erhält den aktuellen Name der Komponente und
REM hängt "_VB" dazu.
sub main (id)
    object_id=Data.GetAttributebyId(id, "relationobject2")
    value = Data.GetAttributebyId(object_id, "name")
    value = value + "_VB"
    call Data.SetAttributebyId(object_id, "name", value)
end sub
```

**JSkript Beispiel**

```
REM Erhält den aktuellen Namen der Komponente und
REM hängt „_JScrip t“ dazu.
function main (id)
{
    object_id=Data.GetAttributebyId(id, "relationobject2");
    value = Data.GetAttributebyId (object_id, "name");
    value = value + "_JScrip t";
    Data.SetAttributebyId(object_id, "name", value);
}
```



**Hinweis:** Vergewissern Sie sich, dass SetAttributebyID immer auf dem richtigen Objekt aufgerufen wird. Zum Beispiel können Sie nicht erwarten, dass der Server den Funktionsaufruf automatisch von einem Subcompitem zur Ergocompbase (relationobject2) überträgt, wenn das Attribut nicht als Subcompitem vorhanden ist.

### 3.1.1.36. SetAttributeRange

#### Syntax

SetOrderAttribute(*bstrAttribute* *name*, *vLB*, *vUB*);

Parameter	Beschreibung
<i>bstrAttribute</i> <i>name</i>	Das Attribut, für welches der Bereich definiert ist.
<i>vLB</i>	Die untere Grenze des Bereichs. Dieser Wert liegt <i>im</i> Bereich.
<i>vUB</i>	Die obere Grenze des Bereichs. Dieser Wert liegt <i>nicht</i> im Bereich.



**Hinweis:** Wenn Bereichsparameter Einheiten-Werte sind, müssen sie als Datenbank-Einheiten weitergereicht werden. Beispielsweise ist die Datenabnakeinheit für die Kategorie Zeit: Sekunden.

Wenn Sie andere Einheiten konvertieren müssen, verwenden Sie bitte die Methoden und Funktionen der [Klasse ScriptItemUnit](#).

#### Rückgabewert

Keiner

#### Beispiel

#### Beispiel

```
sub main(plantype_id)
    plantype_name = Data.GetAttributebyId(plantype_id, "name")
    default_filename="D:\temp\AllInstancesOf" & plantype_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)
        attribute_name = "time"
        lowerBound = 60 'boundary value included
        upperBound = 120 'boundary value excluded
        call Data.SetAttributeRange(attribute_name, lowerBound, upperBound)
        object_id = Data.GetFirstChild(plantype_id, "")
        Do while object_id <> ""
            name = Data.GetAttributebyId(object_id, "name")
            value = Data.GetAttributebyId(object_id, attribute_name)
            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
```

Sie haben unterschiedliche Möglichkeiten Kinderlisten zu erhalten:

- Standard (Entsprechend den Filtereinstellungen im Projekt, nicht sortiert)
- Sortiert (Entsprechend den Filtereinstellungen im Projekt, sortiert), siehe auch die Funktion [SetOrderAttribute](#).
- Ungefiltert (nicht sortiert), siehe auch die Funktion [DisableChildrenListFilter](#).
- **Bereich (ungefiltert, unsortiert, nur Objekte deren Attributwert innerhalb eines vordefinierten Bereichs liegen, werden zurückgegeben).**

Man muss beim Aufruf der Methode [SetAttributeRange](#) den Wertebereich genau definieren. Wenn Sie ein ungültiges Attribut für den Methodenaufruf verwenden (z. B. ein Attribut das nicht in der Kinderliste bzw. nicht an jedem Kind der Kinderliste enthalten ist, oder dem man keinen Wertebereich zuordnen kann) erhalten Sie eine leere Liste.

Bei der Verwendung von `GetFirstChild` und [GetNextChild](#) gelten folgende Einschränkungen:

- Der erste Parameter des Aufrufs muss die ID eines Kindtyps sein, der zweite Parameter wird ignoriert.

```
object_id = Data.GetFirstChild(plantype_id, "")
```

- Es darf kein Projektfiler gesetzt sein.

### 3.1.1.37. [SetAttributesbyId](#)

#### Syntax

```
SetAttributesbyId(bstrObjectId, pVarArrayOfAttrValuePairs);
```

Parameter	Beschreibung
<a href="#">bstrObjectId</a>	Eine Zeichenfolge mit der ID des Objektes.
<a href="#">pVarArrayOfAttrValuePairs</a>	Ein Feld oder ein String, das die zu setzenden Attribut-Werte enthält.

#### Rückgabewert

Kein Rückgabewert

#### Beispiel

#### Beispiel

```
REM Erhält die Attribute über die ID, mit einem Datenbankzugriff.
REM Attribute: 2-dim Feld von Attributen Namen/Werte Paaren
REM Werte: 2-dim Feld von Ergebnissen, dimy ist 1 (attributen / Werte Paar)
REM alle Felder basieren auf Null, z. B. attributes (1,1) bedeutet 2 Elemente sind in einem 2-dimensionalen Feld.
Sub main(id)
    object_id=Data.GetAttributesbyId(id, "relationobject2")
    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
```



**Hinweis:** Vergewissern Sie sich, dass `SetAttributesbyId` immer auf dem richtigen Objekt aufgerufen wird. Zum Beispiel können Sie nicht erwarten, dass der Server den Funktionsaufruf automatisch von einem Subcompitem zur Ergocompbase überträgt, wenn das Attribut nicht als Subcompitem vorhanden ist.



Jscript- Felder sind nicht mit SAFEARRAYs (Visual C++, VB, VBA, VBScript) kompatibel.

Die Information wird implizit als *ein durch Kommas getrennter String* weitergegeben. Daher sind Kommas innerhalb von Attributnamen oder –werten nicht erlaubt, wenn JScript benutzt wird. Geben sie Doppelwerte als „1.23“ statt als „1,23“ an.

**Ausnahme:** Eine ObjektID enthält auch ein Komma, z. B. \$id\$(0:0-1234#0,339). Trotzdem wird dieses Komma als Teil der allgemeinen Sequenz „#0,“ interpretiert.

### 3.1.1.38. SetFetchingSize

#### Syntax

SetFetchi ngSi ze(*ul Fetchi ngSi ze*);

Parameter	Beschreibung
<i>ulFetchingSize</i>	Die Anzahl der Kinder, die mit einem Serverzugriff geholt werden. Standard ist 25.

#### Rückgabewert

kein Rückgabewert

#### Beispiel (Auszug)

#### Beispiel

```
REM Beispiel für commit und rollback einer Transaktion
sub main (object_id)
...
call Data.SetFetchi ngSi ze(50)
...
end sub
```

**3.1.1.39. SetLinkedObjectAttributesbyId****Syntax**

```
SetLinkedObjectAttributesbyId(bstrLinkId,
                               bstrParentObjectId,
                               pVarArrayOfAttrValuePairs);
```

Parameter	Beschreibung
bstrLinkId	Ein String, der die ID des Relationship Objekts enthält.
bstrParentObjectId	Ein String, der die ID eines Vaters der Relation enthält.
pVarArrayOfAttrValuePairs	Ein Feld (oder String), das die zu setzenden Attribut-Wertepaare enthält.

**Rückgabewert**

kein Rückgabewert

**Beispiel****Beispiel**

```
REM Ein Skript, das den Gebrauch von Get/SetLinked Objekt Attributen
REM über die ID aufzeigt.
REM Empfängt die Namen von Objekten (Typ A), die über Relationsnamen
REM auf Objekte (Typ B) verweisen.
REM Um den Code einfach zu halten, wird nur ein Kind abgerufen
```

```
sub main(parent_id)
    relationname="plant_provides_prod"
    parent_base_id=Data.GetAttributebyId(parent_id, "relationobject2")
    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("Skript erfolgreich beendet." )
    else
        MsgBox("Kein relationship Objekt in der Liste <" & relationname & ">." )
    end if
end sub
```

**3.1.1.40. SetLinkedObjectAttributebyId****Syntax**

```
SetLinkedObjectAttributebyId(bstrLinkObjectId,  
                             bstrParentObjectId,  
                             bstrName,  
                             newValue);
```

Parameter	Beschreibung
<i>bstrLinkObjectId</i>	Ein String, der die ID des Relationship Objekts enthält.
<i>bstrParentObjectId</i>	Ein String, der die ID eines Vaters des Relationship Objekts enthält.
<i>bstrName</i>	Ein String, der den Namen des zu setzenden Attributs enthält.
<i>newValue</i>	Eine Variable vom Typ Variant, die den neuen Wert des Attributs enthält.

**Rückgabewert**

Keiner.

**Beispiel:****Beispiel**

```
Rem Ein Skript, das den Gebrauch von Get/SetLinked Objekt Attribute  
über ID aufzeigt.  
Rem Ermittelt die Namen von Objekten (Typ A), die über Relationsnamen  
auf Objekte (Typ B) verweisen.  
Rem Um den Code nicht zu komplex werden zu lassen, wird nur ein Kind  
abgefragt
```

```
Sub main(parent_id)  
    relationname = "plant_provides_prod"  
    parent_base_id = Data.GetAttributebyId(parent_id, "relationobject2")  
    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 ("Keine Relationship- Objekte in der Liste <" & relation-  
        name & ">.")  
    End If  
End Sub
```



**Hinweis:** Benutzen Sie diese Methode, wenn sie ein Objekt vom Typ Relationship und einen seiner Väter kennen, und ein Attribut des anderen Vaters setzen möchten.

**3.1.1.41. ReadSessionData****Syntax**

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

Parameter	Beschreibung
<i>bstrKey</i>	Der Unterschlüssele zu "DELMIA\ergoplan" (Registry) oder "ergoplan" (Datenbank).
<i>bstrAttribute</i>	Der Name des Attributes
<i>bstrLocation</i>	Der Name des Speicherorts

Mögliche Werte für *bstrLocation*

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

**Rückgabewert**

Der (String-) Wert des Attributes, für den der Schlüssel und der Ort bereitgestellt wurden. Wenn die bereitgestellte Schlüssel/Attribut-Kombination nicht existiert, ist der Rückgabewert leer.

**Beispiel****Beispiel**

```

sub main(notused)
subkey_default = "common"
call Dialog.CreatelInputControl ("Subkey", "EditString", "Subkey", subkey_default)
attribute_default = "LanguageId"
call Dialog.CreatelInputControl ("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"

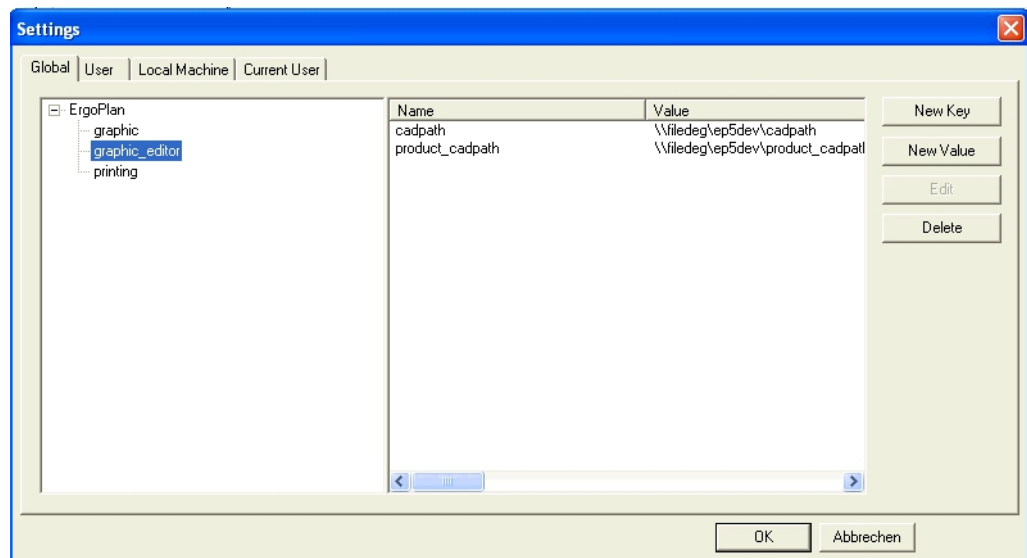
call Dialog.CreatelInputControl ("Location", "ComboBox", "Location", combo)

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")
    attribute=Dialog.GetInputControlValue("Attribute")
    location=Dialog.GetInputControlValue("Location")
    value = Data.ReadSessionData(subkey, attribute, location)
    ...
end if
end sub

```

Die Process Engineer Benutzeroberfläche bietet Zugriff auf den Registrierungseditor über „Werkzeuge/Einstellungen/Wartung“



### 3.1.1.42. WriteSessionData

#### Syntax

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

Parameter	Beschreibung
<i>bstrKey</i>	Der Unterschlüssel zu "DELMIA\ergoplan" (Registry) oder "ergoplan" (Datenbank).
<i>bstrAttribute</i>	Der Name des Attributes.
<i>bstrLocation</i>	Der Name des (Speicher-)Orts
<i>vValue</i>	Der Wert, der gespeichert werden soll.

#### Rückgabewert

Der (String-) Wert des Attributes, für den der Schlüssel und der Ort bereitgestellt wurden. Wenn die bereitgestellte Schlüssel/Attribut-Kombination nicht existiert, ist der Rückgabewert leer.

Mögliche Werte für *bstrLocation*

HKCU	Registry HKEY_CURRENT_USER
HKLM	Registry HKEY_LOCAL_MACHINE
SETTINGSDEFAULT	Registry Defaults HKEY_LOCAL_MACHINE option-configuration
DBUSER	Abhängig vom der PE-Benutzer Datenbank
DBGLOBAL	Global Datenbank

## Beispiel

### Beispiel

```
sub main(notused)
...
call Dialog.CreateInputControl("Subkey", "EditString", "Subkey", subkey)
call Dialog.ModifyInputControl("Subkey", "ReadOnly", True)

call Dialog.CreateInputControl("Attribute", "EditString", "Attribute", attribute)
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, attribute, location, new_value)
end if
end sub
```

## 3.2. Klasse SkriptItemRights

Im Rechtekonzept wurden in der Version PE 5.12 grössere Änderungen vorgenommen. Gruppen-, Benutzer-, Objekt- und- Planungstypenrechte werden jetzt wie Objekte behandelt. So erhalten Sie Informationen zu Rechteobjekten mit Hilfe der bekannten Get-/SetAttributebyId Aufrufen.

Für die Verwendung Ihrer Skripte in der Version 5.12 und später ändern Sie bitte die Skripte, die die bisherigen Rechte Funktionen benutzt haben.

Methode	Beschreibung	Verfügbarkeit
<a href="#">GetUserLogin</a>	Fragt den Login des gegenwärtigen Benutzers ab.	PE 5.6 – 5.11
<a href="#">GetUserFullName</a>	Fragt die Beschreibung (Name) des gegenwärtigen Benutzers ab.	PE 5.6 – 5.11
<a href="#">GetCurrentUser(PE 5.12)</a>	Gibt die Objekt ID des gegenwärtig angemeldeten Benutzers zurück.	<b>Ab PE 5.12</b>
<a href="#">Transfer</a>	Überträgt Objekt- (Planungstypen-) Rechte zwischen zwei Objekten (Planungstypen).	PE 5.10 – 5.11
<a href="#">Transfer (PE 5.12)</a>	Transferiert die Rechte zwischen zwei Objekten oder Planungstypen.	<b>Ab PE 5.12</b>
<a href="#">GetSingleRight</a>	Gibt die Informationen der Rechte-Matrix eines eindeutigen Objekts, Typs oder Funktion zurück.	PE 5.10 – 5.11
<a href="#">GetSingleRight (PE 5.12)</a>	Gibt die Informationen der Rechte-Matrix eines eindeutigen Objekts zurück.	<b>Ab PE 5.12</b>
<a href="#">SetSingleRight (PE 5.12)</a>	Gibt die Informationen der Rechte-Matrix eines eindeutigen Objekts zurück.	<b>Ab PE 5.12</b>
<a href="#">GetAllGroups</a>	Gibt alle Gruppen der Benutzerverwaltung zurück.	PE 5.10 – 5.11
<a href="#">GetAllGroups (PE 5.12)</a>	Gibt alle Gruppen der Benutzerverwaltung zurück.	<b>Ab PE 5.12</b>
<a href="#">GetAllUsers</a>	Gibt alle Benutzer zurück.	PE 5.10 – 5.11
<a href="#">GetAllUsers (PE 5.12)</a>	Gibt alle Benutzer zurück.	<b>Ab PE 5.12</b>
<a href="#">GetUserInfoById</a>	Gibt die Benutzerinformation über die Benutzer-ID zurück.	PE 5.10 – 5.11
<a href="#">GetGroupInfoById</a>	Gibt die Gruppeninformationen über die Gruppen ID zurück.	PE 5.10 – 5.11
<a href="#">GetUserMemberships</a>	Gibt die Gruppenzugehörigkeit(en) zurück.	PE 5.10 – 5.11
<a href="#">GetUserMemberships (PE 5.12)</a>	Gibt die Gruppenzugehörigkeit(en) zurück	<b>Ab PE 5.12</b>
<a href="#">GetGroupMembers</a>	Gibt die Benutzer einer ausgewählten Gruppe zurück.	PE 5.10 – 5.11
<a href="#">GetGroupMembers (PE 5.12)</a>	Gibt die Benutzer einer ausgewählten Gruppe zurück.	<b>Ab PE 5.12</b>
<a href="#">AddRights</a>	Fügt Rechte einem Objekt, einem Planungstypen oder einer Funktion hinzu.	PE 5.10 – 5.11
<a href="#">RemoveRights</a>	Entfernt Rechte eines Objekt, eines Planungstypen oder einer Funktion.	PE 5.10 – 5.11
<a href="#">ConvertRightMaskToArray</a>	Wandelt die Rechte-Matrix in ein Feld um.	<b>Ab PE 5.10</b>
<a href="#">Create</a>	Erzeugt einen Benutzer, eine Gruppe oder ein Funktionsrecht.	<b>Ab PE 5.13</b>
<a href="#">GetFunctionRights</a>	Gibt alle Funktionsrechte zurück.	<b>Ab PE 5.13</b>
<a href="#">GetRightSubjects</a>	Gibt die Zugriffsrechte eines Objektes zurück.	<b>Ab PE 5.16SP4</b>
	Zurücksetzen des Skriptes in seinen Anfangsstatus.	Ab PE 5.10

ResetItem		
-----------	--	--

### 3.2.1. Liste aller XScriptItemRights Methoden

#### 3.2.1.1. **GetUserLogin**

Gültig nur in **PE 5.9 – PE 5.11.**

##### **SYNTAX**

GetUserLogin();

##### **Rückgabewert**

Den Login des Benutzers.

##### **Beispiel**

Diese Funktion ist ab der Version PE 5.12 nicht mehr gültig. Benutzen Sie ab der Version PE 5.12 die Funktion [GetCurrentUser\(PE 5.12\)](#).

##### **Beispiel**

```
REM Empfängt die Benutzer Info
REM Das Script ist unabhängig von der EingangsID
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
```

#### 3.2.1.2. **GetUserFullName**

Gültig nur in **PE 5.9 – PE 5.11.**

##### **SYNTAX**

GetUserFullName();

##### **Rückgabewert**

Der Benutzername

##### **Beispiel**

Diese Funktion ist ab der Version PE 5.12 nicht mehr gültig. Benutzen Sie ab der Version PE 5.12 die Funktion [GetCurrentUser\(PE 5.12\)](#).

##### **Beispiel**

```
REM Empfängt die Benutzer Info
REM Das Script ist unabhängig von der EingangsID
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
```



**3.2.1.3. GetCurrentUser(PE 5.12)****SYNTAX**

GetUserFullName();

**Rückgabewert**

bstrUserId	Die Objekt ID des gegenwärtigen Benutzers.
------------	--

**Beispiel****Beispiel**

```

REM Skript zeigt die Benutzung von Rechten. Transfermethoden
sind gültig ab Version PE 5.12 und später
REM Bedingungen: keine
REM Ergebnis: Zeigt den Login und die Beschreibung des momentan
eingeloggtten Benutzers.
Sub main(notused)
    user_id = Rights.GetCurrentUser()
    if user_id <> "" then
        REM retrieve the login of the user (is stored in the attribute
"nameshort")
        name = Data.GetAttributeById(user_id, "nameshort")
        MsgBox(name)
        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

```

**Hinweis:**

Ab Version PE 5.12 ersetzt diese Funktion die früheren Funktionen *GetUserLogin* und *GetUserFullName*.

**3.2.1.4. Transfer**

**Gültig nur in PE 5.9 – PE 5.11.**

Diese Funktion ist ab der Version PE 5.12 nicht mehr gültig. Benutzen Sie ab der Version PE 5.12 die Funktion [Transfer](#) (PE 5.12).

**SYNTAX**

`Transfer(bstrSourceObjectId, bstrTargetObjectId, bstrRightsDataType);`

Parameter	Beschreibung
<code>bstrSourceObjectId</code>	Die Objekt ID oder die (Planungs)typen GUID.
<code>bstrTargetObjectId</code>	Die Objekt ID oder die (Planungs)typen GUID.
<code>bstrRightsDataType</code>	"Objekt" oder "Typ"

Bitte beachten Sie, dass Sie an die vom Parameter abhängigen richtigen Objekt- oder Typkennungen [bstrRightsDataType](#) weiterreichen müssen.

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

**Beispiel****Beispiel**

```
sub main(notused)
...
REM Beispiel für das Transferieren von Objektrechten
call Rights.Transfer(child1_base_id, child2_base_id, "Object")
...
end sub
```

**3.2.1.5. Transfer (PE 5.12)****SYNTAX**

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

Parameter	Beschreibung
bstrSourceObjectId	Die Objekt ID des Quellobjektes.
bstrTargetObjectId	Die Objekt ID des Zielobjektes.
bstrTransferRightMode	"Clone", "Add" oder "Remove"

**Rückgabewert**

Keiner



**Hinweis:** Bitte beachten Sie, dass Sie an die vom Parameter bstrRightsData-Type abhängigen richtigen Objekt – und Typ IDs weiterreichen müssen

bstrTransferRightMode	Beschreibung
Clone	Kopiert die Rechtematrix von dem Quell- zum Zielobjekt.
Add	Einzelne Zugriffsrechte werden von dem Quell- zum Zielobjekt hinzugefügt.
Remove	Einzelne Zugriffsrechte werden von dem Quell- zum Zielobjekt entfernt.

**Beispiel****Beispiel**

```

Rem Das Skript zeigt die Benutzung von Rechten.
Rem Die Transfermethoden sind gültig ab Version PE 5.12 und später.
Rem Voraussetzungen: Eine einfache Struktur mit einem einzelnen Eltern- und
Rem zwei Kinderknoten.
Rem Ergebnis: Die Rechte werden vom/zum zweiten Kinderknoten kopiert,
Rem hinzugefügt oder entfernt
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", "Transfer 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)
    radi oval = Dialog.GetInputControl Value("3")
    If radi oval = 0 Then
        transfermode = "Clone"
    Else If radi oval = 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, transfer-
mode)
    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, "relationobject2")
End Function

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

### 3.2.1.6. GetSingleRight

Diese Funktion ist ab der Version PE 5.12 nicht mehr gültig. Benutzen Sie ab der Version PE 5.12 die Funktion [GetSingleRight \(PE 5.12\)](#).

#### SYNTAX

`GetSingleRight(IUserId, blsGroup, bstrObjectId, bstrRightsDataType);`

Parameter	Beschreibung
<code>IUserId</code>	Die zu suchende Benutzer- oder Gruppen ID.
<code>blsGroup</code>	Ein Wert, der angibt, ob die <b>IUserId</b> eine user ID (0/False) oder eine group ID (1/True) ist.
<code>bstrObjectId</code>	Die Objekt ID, die Planungstypen (oder Typen) GUID, oder der zu überprüfende Name für das Funktionsrecht (siehe Hinweis).
<code>bstrRightsDataType</code>	"Object", "Type", oder "Funktion" (siehe Hinweis).

#### Rückgabewert

<code>IRightMask</code>	Eine ganze Zahl (integer) (siehe Hinweis).
-------------------------	--

#### Beispiel

#### Beispiel

```
Sub main(i d)
    base_id = Data.GetAttributebyId(i d, "relationobject2")
    users = Rights.GetAllUsers()
    dim users = ubound(users, 1)
    for i = 0 to dim users
        call dispObjectRights(base_id, users(i))
    next
End sub
sub dispObjectRights(i d, useri d)
    userInfo = Rights.GetUserInfoById(useri d)
    rmessage = "User: " & userInfo(0) & vbCRLF & "User-ID: " & cstr(useri d) & vbCRLF & vbCRLF
    rval ue = Rights.GetSingleRight(useri d, 0, i d, "Object")
    ri nfo = Rights.ConvertRightMaskToArray(rval ue)
    di mri nfo = ubound(ri nfo, 1)
    for i = 0 to di mri nfo
        rmessage = rmessage & ri nfo(i, 0) & " --> " & cstr(ri nfo(i, 1)) & vbCRLF
    next
    MsgBox(rmessage)
end sub
```



**Hinweis:** Benutzer und Gruppen-IDs sind ganze Zahlen (integer).

Bitte beachten Sie, dass Sie an die richtigen Objekt- oder Typen-IDs, die vom Parameter [BstrRightsDataType](#) abhängen, weiterreichen müssen.

<b>bstrRightsDataType</b>	<b>Objects</b>	<b>IDs</b>
<a href="#">Object</a>	Normal data objects	POET Object Ids (typisches Format: \$id\$(0:0-8504#0, 339))
<a href="#">Type</a>	Plantypes	GUIDs des <b>slave</b> plantypes (typisches Format: 3e7e6500-5df0-4286-b83e-103685035c14)
<a href="#">Type</a>	Configuration Types	Configuration GUIDs as retrieved using XScriptItemConfig access functions (typical format: 3e7e6500-5df0-4286-b83e-103685035c14)
<a href="#">Funktion</a>	Function rights	Ein String z. B.: "VBA/Open IDE".

**3.2.1.7. GetSingleRight (PE 5.12)****SYNTAX**

GetSingleRight(*bstrObjectId*, *bstrRightSubjectId*);

Parameter	Beschreibung
<i>bstrObjectId</i>	Die Objekt ID, von der die Rechte gelesen werden.
<i>bstrRightSubjectId</i>	Die Objekt ID des Benutzers oder der Gruppe, für die die Rechte der <i>bstrObjectId</i> gelesen werden sollen.

**Rückgabewert**

<i>IRightMask</i>	Eine ganze Zahl (integer).(siehe auch Hinweis)
-------------------	--

**Beispiel****Beispiel**

```

Rem Skript zeigt die Benutzung von Rights.GetAllUsers,
Rem Rights.GetAllGroups, Rights.GetSingleRight und
Rem Rights.ConvertRightMaskToArray Methoden, gültig ab PE 5.12
Rem und später. 2003-09-28, Delmia
Rem Voraussetzungen: Ein einzelner Objektknoten
Rem Legen Sie einige Benutzer an und fügen sie dem Objektknoten
Rem Rechte hinzu, auf dem Sie das Skript testen wollen.
Sub main(id)
    base_id = Data.GetAttributebyId(id, "relationobject2")
    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

```



**Hinweis:** Ist *bstrRightSubjectId* eine leere Zeichenfolge, werden als Ergebnis die Rechte des gegenwärtig angemeldeten Benutzers zurückgegeben.

**3.2.1.8. SetSingleRight (PE 5.12)****SYNTAX**

GetSingleRight(*bstrObjectId*, *bstrRightSubjectId*);

Parameter	Beschreibung
<i>BstrObjectId</i>	Die Objekt ID, an der Rechte vergeben werden sollen.
<i>bstrRightSubjectId</i>	Die Objekt ID des Benutzers oder der Gruppe, für die die Rechte der <i>bstrObjectId</i> gesetzt werden sollen.
<i>IRightMask</i>	Die Rechtematrix für die zu setzenden Rechte.

**Beispiel****Beispiel**

```

sub main(i d)
    base_i d = GetBase(i d)
    comp_name = Data.GetAttributebyId(base_i d, "name")
    user_i d = Rights.GetCurrentUser
    project_i d = GetProject(base_i d)
    project_name = Data.GetAttributebyId(project_i d, "name")

    project_permissions = Rights.GetSingleRight(project_i d, user_i d)
    MsgBox("Transferring the projects <" & project_name & ">permissions to node  
<" & comp_name & ">.")

    call Rights.SetSingleRight(base_i d, user_i d, project_permissions)
end sub

function GetBase(i d)
    GetBase = Data.GetAttributebyId(i d, "relationobject2")
end function

function GetProject(i d)
    GetProject = Data.GetAttributebyId(i d, "ergoproject")
end function

```



**Hinweis:** Ist *bstrRightSubjectId* eine leere Zeichenfolge, werden als Ergebnis die Rechte des gegenwärtig angemeldeten Benutzers zurückgegeben.

**3.2.1.9. GetAllGroups**

Diese Funktion ist ab der Version PE 5.12 nicht mehr gültig. Benutzen Sie ab der Version PE 5.12 die Funktion [GetAllGroups](#) (PE 5.12).

**Syntax**

GetAllGroups();

**Rückgabewert**

<a href="#">pGroupIds</a>	Ein Feld (Reihe/array) von ganzen Zahlen, das alle Gruppen-IDs darstellt.
---------------------------	---

**Beispiel****Beispiel**

```
sub main(notused)
...
groups = Rights.GetAllGroups
dim groups = ubound(groups, 1)
for i = 0 to dim groups
    call GetGroupInfo(groups(i))
next
...
end sub
```



**3.2.1.10. GetAllGroups (PE 5.12)****Syntax**

GetAllGroups();

**Script Return Value**

pGroupIds	Eine Reihe (Feld/array) von ganzen Zahlen, die alle Gruppen-Objekt-IDs darstellen.
-----------	--

**Example****Beispiel**

```
sub main(notused)
...
groups = Rights.GetAllGroups
dim groups = UBound(groups, 1)
for i = 0 to dim groups
  call dispObjectRights(base_id, groups(i))
next
...
end sub
```

siehe auch [ConvertRightMaskToArray](#)

**3.2.1.11. GetAllUsers**

Diese Funktion ist ab der Version PE 5.12 nicht mehr gültig. Benutzen Sie ab der Version PE 5.12 die Funktion [GetAllUsers](#) (PE 5.12).

**Syntax**

GetAllUsers();

**Rückgabewert**

pUserIds	Ein Feld (array) von ganzen Zahlen, das alle Benutzer IDs darstellt.
----------	--

**Beispiel****Beispiel**

```
sub main(notused)
...
users = Rights.GetAllUsers
dim users = ubound(users, 1)
for i = 0 to dim users
  call GetUserInfo(users(i))
next
...
end sub
```

**3.2.1.12. GetAllUsers (PE 5.12)****Syntax**

GetAllUsers();

**Script Return Value**

pUserIds	Ein Feld (Reihe/array) von allen Benutzer Objekt IDs (integer).
----------	---

**Beispiel****Beispiel**

```
sub main(notused)
...
users = Rights.GetAllUsers
dim users = ubound(users, 1)
for i = 0 to dim users
call GetUserInfo(users(i))
next
...
end sub
```

siehe auch  [Beispiel](#)

**3.2.1.13. GetUserInfoByld**

Diese Funktion wird ab Version PE 5.12 nicht mehr verwendet.

**Syntax**

GetUserInfoByld(IUserId);

Parameter	Beschreibung
IUserId	Die Benutzer ID.

**Rückgabewert**

pUserInfo	Ein eindimensionales Feld (Wertepaar), dass den Benutzernamen (login) / und die Beschreibung (user full name) enthält.
-----------	--

**Beispiel****Beispiel**

```
...
sub GetUserInfo(userId)
userInfo = Rights.GetUserInfoByld(userId)
dim userInfo = ubound(userInfo, 1)
if dim userInfo < 0 then
message = "User does not exist (invalid user ID " & userId & " )"
else
message = "User: " & userInfo(0) & vbTab & "User-ID: " & cstr(userId)
end if
ts.WriteLine(message)
end sub
...
```

**3.2.1.14. GetGroupInfoById**

Diese Funktion wird ab Version PE 5.12 nicht mehr verwendet.

**Syntax**

GetGroupInfoById(*GroupId*);

Parameter	Beschreibung
<i>GroupId</i>	Die Gruppen ID.

**Rückgabewert**

<i>pGroupInfo</i>	Ein eindimensionales Feld (Wertepaar), dass den Gruppenname (login) / und die Beschreibung (user full name) enthält.
-------------------	--

**Beispiel****Beispiel**

```
...
Sub GetGroupInfo(groupId)
    groupInfo = Rights.GetGroupInfoById(groupId)
    dim groupInfo = ubound(groupInfo, 1)
    if dim groupInfo < 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
...
```

**3.2.1.15. GetUserMemberships**

Diese Funktion ist ab der Version PE 5.12 nicht mehr gültig. Benutzen Sie ab der Version PE 5.12 die Funktion [GetUserMemberships \(PE 5.12\)](#).

**Syntax**

GetUserMemberships(*UserId*);

Parameter	Beschreibung
<i>UserId</i>	Die Benutzer ID.

**Rückgabewert**

<i>pGroupIds</i>	Eine Reihe von ganzen Zahlen, in dem alle Gruppen IDs aufgeführt werden, zu dem ein Benutzer gehört.
------------------	--

**Beispiel****Beispiel**

```
sub main(notused)
...
    usermemberships=Rights.GetUserMemberships(users(i))
    dim usermemberships = ubound(usermemberships, 1)
    if dim usermemberships < 0 then
        ts.WriteLine("Not a member of a special group.")
    end if
    for j = 0 to dim usermemberships
        call GetGroupInfo(usermemberships(j))
    next
...
end sub
```

**3.2.1.16. GetUserMemberships (PE 5.12)****Syntax**

GetUserMemberships([bstrGroupObjectId](#));

Parameter	Beschreibung
<a href="#">bstrGroupObjectId</a>	Die Benutzer ID.

**Rückgabewert**

<a href="#">pGroupIds</a>	Eine Reihe von ganzen Zahlen, in dem alle Gruppen Objekt IDs aufgeführt werden, zu dem ein Benutzer gehört.
---------------------------	---

**Beispiel****Beispiel**

```
sub main(notused)
...
usermemberships=Rights.GetUserMemberships(users(i))
dim usermemberships = UBound(usermemberships, 1)
next
...
end sub
```

siehe auch: [GetGroupMembers \(PE 5.12\)](#)



**Hinweis:** Die Syntax des Funktionsaufrufs hat sich im Vergleich zu dem Funktionsaufruf in der Version 5.11 nur wenig geändert, jedoch der übergebene Parameter. Der übergebene Parameter ist ein String einer Objekt ID und das zurückgegebene Feld besteht aus Objekt-IDs (POET typisches Format: " \$id \$ ( 0:0-8504 # 0, 339 ) " ), anstelle der früher gebrauchten ganzen Zahlen (0, 1, ..., n).

### 3.2.1.17. **GetGroupMembers**

Diese Funktion ist ab der Version PE 5.12 nicht mehr gültig. Benutzen Sie ab der Version PE 5.12 die Funktion [GetGroupMembers \(PE 5.12\)](#).

#### **Syntax**

**GetGroupMembers**(I GroupId);

Parameter	Beschreibung
IGroupId	Die Gruppen ID.

#### **Rückgabewert**

pUserIds	Eine Liste von ganzen Zahlen, die alle Benutzer-IDs darstellen, die zur Gruppe IGroupId gehören.
----------	--

#### **Beispiel**

#### **Beispiel**

```
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
```

**3.2.1.18. GetGroupMembers (PE 5.12)****Syntax**

GetGroupMembers(*bstrGroupObjectId*);

Parameter	Beschreibung
<i>bstrGroupObjectId</i>	Die ID der Gruppe.

**Rückgabewert**

<i>pUserIds</i>	Eine Liste von Strings, die alle Benutzer Objekt-IDs darstellen, die zur Gruppe <i>bstrGroupObjectId</i> gehören.
-----------------	---

**Beispiel****Beispiel**

REM Script zeigt die Benutzung von Rechten. Die Transfermethode ist ab PE 5.12 und später gültig.  
 REM Voraussetzungen: Wenigstens einige Benutzer und einige Gruppen, mit denen sie verknüpft sind.  
 REM Ergebnis: Eine Textdatei, die die Benutzer und Gruppen, die in der Datenbank enthalten sind, genauso wie deren Mitglieder bzw. Mitgliedschaften anzeigt.

```
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
  Next
End Sub
```

```
        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
```



**Hinweis:** Die Syntax des Funktionsaufrufs hat sich im Vergleich zu dem Funktionsaufruf in der Version 5.11 nur wenig geändert, jedoch der übergebene Parameter. Der übergebene Parameter ist ein String einer Objekt ID und das zurückgegebene Feld besteht aus Objekt-IDs (POET typisches Format: " \$id \$ ( 0:0-8504 # 0, 339 ) "), anstelle der früher gebrauchten ganzen Zahlen (0, 1, ..., n).

**3.2.1.19. AddRights****Syntax**

AddRights(*bstrObjectId*, *bstrRightsDataType*, *pRightsInfo*);

Diese Funktion ist ab der Version PE 5.12 nicht mehr gültig. Benutzen Sie ab der Version PE 5.12 die Funktion [GetSingleRight](#) (PE 5.12).

Parameter	Beschreibung
<i>bstrObjectId</i>	Ein String, der die ID des Objekts enthält.
<i>bstrRightsDataType</i>	"Object" (für Objekte) oder "Type" (für Planungstypen und Konfigurationstypen)
<i>pRightsInfo</i>	Die Struktur enthält die hinzuzufügenden Rechteinformationen (siehe auch Hinweis).

**Beispiel****Beispiel**

```

sub main(id)
    base_id = Data.GetAttributeById(id, "relationobject2")
    user_id = 5
    rightmask = 782 ("Change")
    Dim rightinfo(0,3)
    rightinfo(0,0) = user_id
    rightinfo(0,1) = 0
    rightinfo(0,2) = rightmask
    rightinfo(0,3) = 0

    call Rights.AddRights(base_id, "Object", rightinfo)
    ...
end sub

```



**Hinweis:** Die Rechtematrix ist eine ( $N \times 4$ ) dimensionale Matrix.  $N$  bedeutet, dass Sie eine beliebige Anzahl von Benutzern und Gruppen in einem einzelnen Anruf ermitteln können. Das Wertepaar (0,0) -- (0,3) stellt den ersten Benutzer (Gruppe) dar, (1,0) -- (1,3) den zweiten Benutzer (Gruppen), usw...

Die Struktur ist folgendermaßen organisiert:

Element	Inhalt	Hinweis
<b>0</b>	User ID	
<b>1</b>	IsGroup Flag	0: User 1: Group
<b>2</b>	Right Mask	Eine ganze Zahl, die die Einstellungen der <a href="#">Rechtematrix</a> darstellt.
<b>3</b>	IsOwner Flag	0: Benutzer ist <b>nicht</b> Besitzer des Objektes. 1: Benutzer ist Besitzer des Objektes.

Das Hinzufügen von Rechten entfernt nicht die gegenwärtigen Rechte, sondern fügt sie einzeln zu den gegenwärtigen Rechten hinzu. Wenn Sie die gegenwärtigen Rechte *ersetzen* wollen, sollten Sie [RemoveRights](#) aufrufen, wo Sie rightmask = 1023 setzen und erst dann AddRights aufrufen.



**3.2.1.20. RemoveRights**

Diese Funktion ist ab der Version PE 5.12 nicht mehr gültig. Benutzen Sie ab der Version PE 5.12 die Funktion [SetSingleRight](#) (PE 5.12).

**Syntax**

```
RemoveRights([bstrObjectId, bstrRightsDataType,
pRightsInfo);
```

Parameter	Beschreibung
<a href="#">bstrObjectId</a>	Ein String, der die ID des Objekts enthält.
<a href="#">bstrRightsDataType</a>	"Object" (für Objekte) oder "Type" (für plantypes und configuration types)
<a href="#">pRightsInfo</a>	Die Struktur enthält die hinzuzufügenden Rechteinformationen (siehe auch Hinweis).

**Rückgabewert**

Keiner

**Beispiel****Beispiel**

```
sub main(id)
    base_id = Data.GetAttributeById(id, "relationobject2")
    user_id = 5
    rightmask = 782 ("Change")

    Dim rightinfo(0, 3)
    rightinfo(0, 0) = user_id
    rightinfo(0, 1) = 0
    rightinfo(0, 2) = rightmask
    rightinfo(0, 3) = 0

    Call Rights.RemoveRights(base_id, "Object", rightinfo)
    ...
end sub
```



**Hinweis** : Die Rechematrix ist eine (N X 4) dimensionale Matrix. N bedeutet, dass Sie eine beliebige Anzahl von Benutzern und Gruppen in einem einzelnen Anruf ermitteln können. Das Wertepaar (0,0) -- (0,3) stellt den ersten Benutzer (Gruppe) dar, (1,0) -- (1,3) den zweiten Benutzer (Gruppen), usw...

Die Struktur ist folgendermaßen organisiert:

Element	Inhalt	Hinweis
<b>0</b>	<a href="#">User ID</a>	
<b>1</b>	<a href="#">IsGroup Flag</a>	0: User 1: Group
<b>2</b>	<a href="#">Right Mask</a>	Eine ganze Zahl, die die Einstellungen der <a href="#">Rechematrix</a> darstellt.
<b>3</b>	<a href="#">IsOwner Flag</a>	0: Benutzer ist <b>nicht</b> Besitzer des Objektes. 1: Benutzer ist Besitzer des Objektes.

Um alle Rechte zu entfernen, muss 1023 (alle Werte werden auf Null gesetzt) gesetzt werden. Mit 1024 werden Benutzer oder Gruppen von Objekten oder Planungstypen- oder Typenrechtsliste entfernt. In diesem Fall werden die Rechte von einer übergeordneten Ebene erhalten.

**3.2.1.21. ConvertRightMaskToArray****Syntax**

ConvertRightMaskToArray([IRightMask](#));

Parameter	Beschreibung
<a href="#">IRightMask</a>	Edit

**Rückgabewert**

<a href="#">*pRightsArray</a>	Edit
-------------------------------	------

**Beispiel für PE 5.10 und PE 5.11 (funktioniert nicht in PE 5.12 und höher)**

**Beispiel**

```

sub main(id)
    base_id = Data.GetAttributeById(id, "relationobject2")
    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))
    next
    MsgBox(rmessage)
end sub

```

**Beispiel****Beispiel für PE 5.12**

Siehe auch: [ConvertRightMaskToArray](#)

Die Rechtematrix ist folgendermaßen organisiert.

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

## 3.2.1.22. Create

## Syntax

Create(**bstrObjectType**);

Parameter	Beschreibung
<b>bstrObjectType</b>	Das zu erzeugende Objekt. Dies kann ein Benutzer (User), eine Gruppe (Group) oder ein Funktionsrecht (Function) sein.

## Rückgabewert

<b>bstrRightsObjectId</b>	Das Objekt, welches erzeugt wurde. Dies kann ein Benutzer, eine Gruppe oder ein Funktionsrecht sein.
---------------------------	--

## Beispiel

## Beispiel

```
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")
    'PE5.14 and previous versions
    'call Data.SetAttributebyId(new_user_id, "password", "fgb")
    'PE5.15 und spätere Versionen - modification of script exam-
    ple required before use! Refer to remarks.
    call Data.SetAttributebyId(new_user_id, "password",
    "c05651bcf32bd68259c0862d9f10cea9b0c14bcc")
    MsgBox("User Created successfully.")
end if
...
end sub
```



## Hinweis

*Kennwortverschlüsselung (Password Encryption):*

Öffnen Sie die Eingabeaufforderung (DOSBox) und tragen folgende Befehlszeile ein:

```
C:\temp> simplecryptadmin -hs <String> SHA1
```

(<String>: Ist das zu verschlüsselnde Kennwort - ohne Anführungszeichen)

Für das obere Beispiel würde die Befehlszeile dann lauten:

```
C:\temp> simplecryptadmin -hs fgb SHA1
```

► Das Ergebnis sieht folgendermaßen aus:

```
String Hash: c05651bcf32bd68259c0862d9f10cea9b0c14bcc
Operation successfully done
```

Im Skript ersetzen Sie nun das Kennwort in der Zeile:

```
call Data.SetAttributebyId(new_user_id, "password", "fgb")
```

durch das verschlüsselte Passwort.

```
call Data.SetAttributebyId(new_user_id, "password",
"c05651bcf32bd68259c0862d9f10cea9b0c14bcc")
```



**Hinweis:** Neue Benutzer, Gruppen oder Funktionsrechte müssen richtig initialisiert werden, um auf der Oberfläche sichtbar zu sein. Verwenden Sie deshalb die Funktion Create sehr vorsichtig: Falls Sie Bedenken bei der Handhabung der Funktion haben, setzen Sie sich bitte mit unserer Anwender-Hotline in Verbindung:

Telefon: +49 / 711 / 27 300-400;

Telefax: +49 / 711 / 27 300-599 oder

E-Mail: [support@delmia.de](mailto:support@delmia.de)

### 3.2.1.23. GetFunctionRights

#### Syntax

GetFunctionRights();

Parameter	Beschreibung
keine	-

#### Rückgabewert

pFunctionIds	Ein Feld mit allen Funktionsrechten.
--------------	--------------------------------------



**Hinweis:** Das zurückgegebene Feld der Funktionsrechte ist flach, enthält aber hierarchische Informationen, da Funktionsrechte in einer Baumstruktur organisiert sind. Jedes Funktionsrecht hat ein Attribut „rootfunctionname“. Wenn dieses Attribut leer ist, ist das Funktionsrecht ein Top-Level Funktionsrecht.

#### Beispiel

#### Beispiel

```
sub main(notused)
  users = Rights.GetAllUsers()
  dim users = ubound(users, 1)
  functionname="Copy Project"

  for i = 0 to dim users
    call dispFunctionRights(functionname, users(i))
  next
end sub

sub dispFunctionRights(id, user id)
  userinfo = Rights.GetUserInfoById(user id)
  rmessage = "User: " & userinfo(0) & vbCrLf & "User-ID: " & cstr(user id) & vbCrLf & vbCrLf
  rval ue = Rights.GetSingleRight(user id, 0, id, "Function")
  rinfo = Rights.ConvertRightMaskToArray(rval ue)
  dim rinfo = ubound(rinfo, 1)
  for i = 0 to dim rinfo
    rmessage = rmessage & rinfo(i, 0) & " --> " & cstr(rinfo(i, 1))
  next

  MsgBox(rmessage)
end sub
```

### 3.2.1.24. GetRightSubjects

#### Syntax

GetRightSubjects(*bstrObjectId*);

Parameter	Beschreibung
<i>bstrObjectId</i>	The object ID from which the permission are to be read.

#### Rückgabewert

<i>lRightMask</i>	An integer representing a bit mask (see remarks).
-------------------	---



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

#### Beispiel

#### Beispiel

```
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
```

### 3.3. Klasse ScriptItemGrid

Methode	Beschreibung
<b>Create</b>	Erstellt ein Formular (Sheet). Eine wesentliche Anforderung an Kalkulationen ist die Darstellung der Ergebnisse in Formularen. Formulare (Stingray Objective Grids) können per Script erstellt werden. Dazu dienen die Methoden Create und SetCell. Mit <b>rows</b> , <b>cols</b> , <b>header</b> wird ein Formularname mit rows (Zeilen) und cols (Spalten) erzeugt. Die Spaltenbezeichnungen werden in header zusammengefasst (durch pipe-Zeichen getrennt).
<b>SetCell</b>	Setzt den Wert einer bestimmten Zelle im Formular. Wenn ein Formular verwendet werden soll, dann muss bei den Eigenschaften des Skriptes die CheckBox <Use sheet> bzw. „Formular verwenden“ aktiviert werden.
<b>ResetItem</b>	Zurücksetzen des Skriptes in seinen Anfangsstatus.

#### 3.3.1. Liste aller XScriptItemGrid Methoden

##### 3.3.1.1. Create

###### SYNTAX

Create(*bstrCaption*, *ulRows*, *ulCols*, *bstrColumnHeaders*);

Parameter	Beschreibung
<i>bstrCaption</i>	Die Überschrift des Formulars.
<i>ulRows</i>	Die Anzahl der Zeilen des Formulars.
<i>ulCols</i>	Die Anzahl der Spalten des Formulars.
<i>bstrColumn-headers</i>	Die Spaltenüberschrift. Die einzelnen Überschriften werden mit ' ' (pipe) getrennt.

###### Rückgabewert

kein Rückgabewert

###### Beispiel

##### Beispiel

```
REM Zeigt ein Formular
REM Das Skript ist unabhängig von der Eingangs ID
sub main (id)
  call Sheet.Create("My Sheet", 10, 3, "Col 1|Col 2|Col 3")
  call Sheet.SetCell(1, 1, "1000")
  call Sheet.SetCell(5, 1, "5000")
end sub
```



**Hinweis:** Es ist notwendig den Kontrollkästchen „Display sheet“ im Eigenschaftendialog vom Skript zu überprüfen um ein Client Fenster zu initialisieren, dass das Raster hält. Zur Zeit ist die Klasse XscriptItemGrid nur zu Betrachtungszwecken da und hat nur einfache Funktionen.

### 3.3.1.2. SetCell

#### SYNTAX

SetCell (ulRow, ulCol, vValue);

Parameter	Beschreibung
ulRow	Der Zeilenindex der Zelle. Ganz Oben ist 1.
ulCol	Der Spaltenindex der Zelle. Ganz links ist 1.
vValue	Der Wert, der in der Zelle angezeigt werden soll.

#### Rückgabewert

kein Rückgabewert

#### Beispiel

### Beispiel

```
REM Zeigt ein Formular
REM Das Skript ist unabhängig von der Eingangs ID
sub main (id)
  call Sheet.Create("My Sheet", 10, 3, "Col 1|Col 2|Col 3")
  call Sheet.SetCell(1, 1, "1000")
  call Sheet.SetCell(5, 1, "5000")
end sub
```

### 3.4. Klasse ScriptItemGraphic

Zugriffe auf Grafiken	
Methoden	Beschreibung
<b>CreateBitmap</b>	Erstellt ein Bitmap der zugehörigen 3D Grafikszenen.
<b>CreateContextBitmap</b>	Erstellt ein Bitmap der zugehörigen 3D Grafikszenen im Zusammenhang mit den Nachbarelementen.
<b>CreateFile</b>	Erstellt eine Image Datei (*.bom oder *.objekt Format), eines Grafikobjekts.
<b>CreatePreviewGraphic</b>	Erzeugt eine Grafikdatei aus PE-Grafiken. Diese Datei wird in dem Verzeichnis ... <i>DELMIA\PPRClient\preview_graphic</i> abgelegt und wird von DPM V5 als Vorschau Grafik verwendet. Das graphische Format von DPE kann nicht von DPM gelesen werden.
<b>ShowGraphic</b>	Method for the creation of a graphic view.
<b>GetBBoxSize</b>	Holt die Bounding Box des zugehörigen 3D Grafikelements.
<b>GetGraphicMatrix</b>	Holt die 4x4 Grafik Matrix (Rotationen und Translationen)
<b>GetGraphicRGB</b>	Holt die RGB Farbwerte eines Grafikobjekts.
<b>GetGraphicAlpha</b>	Holt den alpha Wert eines Grafikobjekts.
<b>GetTranslation</b>	Holt die Translation eines Grafikobjekts.
<b>GetRotation</b>	Holt die Rotation eines Grafikobjekts.
<b>SetGraphicMatrix</b>	Setzt die 4x4 Grafik Matrix (Rotationen und Translationen)
<b>SetGraphicRGB</b>	Setzt die RGB Farbwerte eines Grafikobjekts.
<b>SetGraphicAlpha</b>	Setzt den alpha Wert eines Grafikobjekts.
<b>SetTranslation</b>	Setzt die Translation eines Grafikobjekts.
<b>SetRotation</b>	Setzt die Rotation eines Grafikobjekts.
<b>ResetItem</b>	Zurücksetzen des Skriptes in seinen Anfangsstatus.



### 3.4.1. Liste aller XScriptItemGraphic Methoden

#### 3.4.1.1. CreateBitmap

##### SYNTAX

```
CreateBitmap(bstrObjectId,
             bstrFileName,
             bstrSettings,
             dblSizeA,
             dblSizeB);
```

Parameter	Beschreibung
bstrObjectId	Ein String, der die ID des Objekts enthält.
bstrFileName	Der Name der Datei in die das Bitmap gespeichert werden soll.
bstrSettings	Definiert die Grafikeinstellungen.
dblSizeA	Die Breite des Bitmaps. Die Werte sind zu einem Integer gerundet (Variablen vom Datentyp <b>Integer</b> werden als 16-Bit-Zahlen (2 Bytes) in einem Bereich von -32.768 bis 32.767 gespeichert).
dblSizeB	Die Höhe des Bitmaps. Die Werte sind zu einem Integer gerundet.

##### Rückgabewert

kein Rückgabewert

##### Beispiel (Auszug)

#### Beispiel

```
sub main(id)
...
    camposx=0.0
    camposy= round(((ymax(i)+ymid(i)) / 2), 0)
    camposz= round(((zabsmax + zabsmin) / 2), 0)

    width=0.0
    height = zabsmax - zabsmin
    scale = 768/height

    bmpsizey=scale * (ymax(i)-ymid(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
```



**Hinweis:** Wenn der *bstrSettings* = "", dann beinhalten die Bilder die Standard 3D-Sicht. Variablen vom Datentyp Double (Gleitkommazahl mit doppelter Genauigkeit) werden als 64-Bit-Gleitkommazahlen (8 Bytes) nach IEEE im Bereich von -1,79769313486231E308 bis -4,94065645841247E-324 für negative Werte und von 4,94065645841247E-324 bis 1,79769313486232E308 für positive Werte gespeichert.

### 3.4.1.2. CreateContextBitmap

#### SYNTAX

```
CreateContextBitmap(bstrObjectId,
                    bstrContextObjectId,
                    bstrFileName,
                    bstrSettings,
                    dblSizeA,
                    dblSizeB);
```

Parameter	Beschreibung
bstrObjectId	Ein String, der die ID des Objekts enthält.
bstrContextObjectId	Ein String, der die ID des Kontext Objekts enthält.
bstrFileName	Der Name der Datei, in die das Bitmap gespeichert werden soll.
bstrSettings	Definiert die Grafikeinstellungen.
dblSizeA	Die Breite des Bitmaps. Die Werte sind zu einem Integer gerundet.
dblSizeB	Die Höhe des Bitmaps. Die Werte sind zu einem Integer gerundet.

#### Rückgabewert

kein Rückgabewert

#### Beispiel (Ein Auszug)

#### Beispiel

```
sub main(i d)
...
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
bmpsizex=scale * (ymax(i)-ymin(i))
bmpsizex=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, bmpsizex, bmpsizex)
...
end sub
```



**Hinweis:** Wenn der *bstrSettings* = "", dann beinhalten die Bilder die Standard 3D-Sicht.

**3.4.1.3. GetBBoxSize****SYNTAX**

GetBBoxSize(**bstrObjectId**);

Parameter	Beschreibung
<b>bstrObjectId</b>	Ein String, der die ID des Objekts enthält.

**Rückgabewert**

Die Koordinaten der 3x2 Matrix geben die Position der Bounding Box an. Die Anordnung ist folgendermaßen:

<b>0</b>	<b>1</b>	<b>=</b>	<b>x<sub>min</sub></b>	<b>x<sub>max</sub></b>
<b>2</b>	<b>3</b>		<b>y<sub>min</sub></b>	<b>y<sub>max</sub></b>
<b>4</b>	<b>5</b>		<b>z<sub>min</sub></b>	<b>z<sub>max</sub></b>

**Beispiel****Beispiel**

```
sub main (id)
...
bboxsize = Graphic.GetBBoxSize(chlid_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
```

**3.4.1.4. CreateFile****SYNTAX**

CreateFile([bstrObjectId](#), [bstrFileNameWithNoExtension](#))

Parameter	Beschreibung
<a href="#">bstrObjectId</a>	Ein String, der die ID des Objekts enthält.
<a href="#">bstrFileNameWithNoExtension</a>	Sie können entweder eine Platzhalter und einen Dateinamen ohne Erweiterung angeben, oder Sie geben eine Standard-Dateierweiterung an. Sehen Sie auch die nachfolgende Tabelle und das Beispiel darunter.

<b>b</b>	Erzeugt eine DPE Grafik mit der Erweiterung <b>.bom</b>
<b>o</b>	Erzeugt eine E4 Grafik mit der Erweiterung <b>.objekt</b> . Dieses Format kann nur zum Import in ERGOPLAN 4.xx verwendet werden.
<b>d</b>	Erzeugt eine AutoCAD (DXF) Export-Grafik mit der Erweiterung <b>.dxf</b>
<b>v</b>	Erzeugt eine VRML1 Grafik mit der Erweiterung <b>.wrl</b>
<b>c</b>	Erzeugt eine CATIA Tesselated Format (CGR) Grafik mit der Erweiterung <b>.cgr</b>

Wenn kein Verzeichnis angegeben wird, wird das Standardverzeichnis (Xbof-graph\_koerper) gewählt.

**Rückgabewert**

Keiner

**Beispiel****Beispiel**

```
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
```

### 3.4.1.5. CreatePreviewGraphic

#### SYNTAX

CreatePreviewGraphic (bstrObjectId, bstrType, dblDeep, dblOverwrite, bUseExisting);

Parameter	Beschreibung
bstrObjectId	Ein String, der die ID des Objekts enthält.
bstrType	Ein String, der die Dateierweiterung der erzeugten Grafik angibt. Unterstützt werden: CGR: "cgr", VRML: "vrl" und AutoCAD DXF: "dxf".
dblDeep	Angabe ob die Grafik nur von dem Knoten auf dem das Skript aufgerufen wird erzeugt werden soll (0) oder auch alle Grafiken der Kinder berücksichtigt werden sollen (1).
dblOverwrite	Mit diesem Parameter wird festgelegt ob eine bereits existierende Grafikdatei (mit dem gleichen Namen und derselben Dateierweiterung) überschrieben werden darf (1) oder nicht (0).
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.

#### Rückgabewert

Keiner

#### Beispiel

#### Beispiel

```
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 entry the cadpath and then the product\_cadpth setting is used, for resources and products respectively. If none of them are no file will be 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.

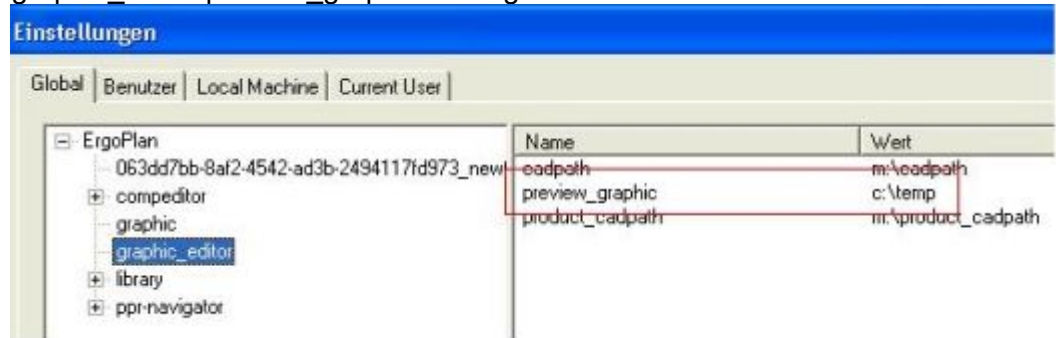
**Note:** DPE does not use the created files. There are exclusively for DPM.

- 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.

Purpose: Creates a graphic file, representing the geometry of the object, including geometries of children.

The file is created in a special directory, defined with the 'graphic\_editor/preview\_graphic' setting:



Implemented especially for V5 users.

The preview graphic of an object is intended to be used in V5 to accelerate (pre)viewing of E5 graphic. It's up to the user to keep the preview graphic files up to date. The created files are not used in E5.

### 3.4.1.6. ShowGraphic

Method zum Erzeugen einer Grafikanzeige.

#### SYNTAX

ShowGraphic (bstrObjectId, bstrSettings);

Parameter	Beschreibung
bstrObjectId	Ein String, der die ID des Objekts enthält.
bstrSettings	Definiert die Grafikeinstellungen. 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] Abstand der Gitterlinien ist 200mm [GRIDCOL:255] Die Farbe der Gitterlinie ist Rot

#### Rückgabewert

Keiner.

#### Beispiel

#### Beispiel

```
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
```

**3.4.1.7. GetGraphicMatrix****SYNTAX**

GetGraphicMatrix(**BSTR** bstrObjectId);

Parameter	Beschreibung
bstrObjectId	Ein String, der die ID des Objekts enthält.

**Rückgabewert**

Ein Feld für die rotation/translation Matrix des Grafikobjektes.

r <sub>11</sub>	r <sub>12</sub>	PE13	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

**Beispiel****Beispiel**

REM Manipulation der Position eines graphischen Objektes (nur Translation)

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

### 3.4.1.8. GetGraphicRGB

#### SYNTAX

GetGraphicRGB([bstrObjectId](#));

Parameter	Beschreibung
<a href="#">bstrObjectId</a>	Ein String, der die ID des Objekts enthält.

#### Rückgabewert

Ein Feld mit der Angabe der RGB Werte des Objektes.

R	0
G	1
B	2

#### Beispiel

### Beispiel

```
sub main (i d)
  Dim rgb
  rgb = Graphic.GetGraphicRGB(i d)
  r=rgb(0)
  g=rgb(1)
  b=rgb(2)

  rgb(0)=100
  rgb(1)=100
  rgb(2)=100
  Call Graphic.SetGraphicRGB(i d, rgb)
end sub
```



**3.4.1.9. GetGraphicAlpha****SYNTAX**

GetGraphi cAl pha(BSTR bstrObj ectI d);

Parameter	Beschreibung
bstrObjectId	Ein String, der die ID des Objekts enthält.

**Rückgabewert**

Das „Alpha“ vom Objekt (Inverse Transparenz).

**Beispiel****Beispiel**

```
sub main (i d)
    al pha = Graphi c. GetGraphi cAl pha(i d)
    al pha = al pha/2
    cal l Graphi c. SetGraphi cAl pha(i d, al pha)
end sub
```

**3.4.1.10. GetTranslation****SYNTAX**

GetTransl ati on(BSTR bstrObj ectI d);

Parameter	Beschreibung
bstrObjectId	Ein String, der die ID des Objekts enthält.

**Rückgabewert**

Ein Feld von 3 Wertepaaren (Radianten), das die Rotation (basierend auf dem Ursprung) vom zugehörigen grafischen Objekt zurückgibt.

x	0
y	1
z	2

**Beispiel****Beispiel**

```
sub main (i d)
    Di m posi ti on
    posi ti on = Graphi c. GetTransl ati on(i d)
    x=posi ti on(0)
    y=posi ti on(1)
    z=posi ti on(2)
    posi ti on(0)=500
    posi ti on(1)=1500
    posi ti on(2)=2500
    cal l Graphi c. SetTransl ati on(i d, posi ti on)
end sub
```

**3.4.1.11. GetRotation****SYNTAX**

GetRotation(**BSTR bstrObjectId**);

Parameter	Beschreibung
<b>bstrObjectId</b>	Ein String, der die ID des Objekts enthält.

**Rückgabewert**

Ein Feld von 3 Wertepaaren (Radianten), das die Rotation (basierend auf dem Ursprung) vom zugehörigen graphischen Objekt zurückgibt.

x	0
y	1
z	2

**Beispiel****Beispiel**

```
sub main (i d)
  Dim angle
  angle = Graphic.GetRotation(i d)
  x=angle(0)
  y=angle(1)
  z=angle(2)
  angle(0)=1.5
  angle(1)=1.5
  angle(2)=1.5
  call Graphic.SetRotation(i d, angle)
end sub
```

**3.4.1.12. SetGraphicMatrix****SYNTAX**

SetGraphicMatrix(**BSTR** bstrObjectId, **pMatrix**);

Parameter				Beschreibung			
bstrObjectId				Ein String, der die ID des Objekts enthält.			
pMatrix				Ein Feld von 16 Werten, das die Rotations-/ Transformationsmatrix des zugehörigen grafischen Objektes zurückgibt.			
r <sub>1</sub>	r <sub>12</sub>	PE13	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

**Rückgabewert**

keiner

**Beispiel****Beispiel**

```

REM Verändert die Position des Grafikobjektes (nur Translation)
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

```

**3.4.1.13. SetGraphicRGB****SYNTAX**

SetGraphicRGB(*bstrObjectId*, *pRGB*);

Parameter	Beschreibung
<i>bstrObjectId</i>	Ein String, der die ID des Objekts enthält.
<i>pRGB</i>	Ein Feld von 3 Werten, das die RGB Werte des zugehörigen grafischen Objektes zurückgibt.

R	0
G	1
B	2

**Rückgabewert**

keiner

**Beispiel****Beispiel**

```
sub main (i d)
  Dim rgb
  rgb = Graphic.GetGraphicRGB(i d)
  r=rgb(0)
  g=rgb(1)
  b=rgb(2)

  rgb(0)=100
  rgb(1)=100
  rgb(2)=100
  Call Graphic.SetGraphicRGB(i d, rgb)
end sub
```

**3.4.1.14. SetGraphicAlpha****SYNTAX**

SetGraphicAlpha(*bstrObjectId*, *dblAlpha*);

Parameter	Beschreibung
<i>bstrObjectId</i>	Ein String, der die ID des Objekts enthält.
<i>dblAlpha</i>	Setzt einen neuen Wert von „alpha“ (inverse Transparenz).

**Rückgabewert**

keiner

**Beispiel****Beispiel**

```
sub main (i d)
  alpha = Graphic.GetGraphicAlpha(i d)
  alpha = alpha/2
  Call Graphic.SetGraphicAlpha(i d, alpha)
end sub
```

**3.4.1.15. SetTranslation****SYNTAX**

SetTransl ati on(bstrObj ectI d, dbl Posi ti on);

Parameter	Beschreibung
bstrObjectId	Ein String, der die ID des Objekts enthält.
dblPosition	Ein Feld von 3 Werten, die zur Angabe der neuen Translationswerte gebraucht werden.

x	0
y	1
z	2

**Rückgabewert**

keiner

**Beispiel****Beispiel**

```
sub main (i d)
  Dim posi ti on
  posi ti on = Graphi c. GetTransl ati on(i d)
  x=posi ti on(0)
  y=posi ti on(1)
  z=posi ti on(2)
  posi ti on(0)=500
  posi ti on(1)=1500
  posi ti on(2)=2500
  call Graphi c. SetTransl ati on(i d, posi ti on)
end sub
```

**3.4.1.16. SetRotation****SYNTAX**

**SetRotation**(*bstrObjectId*, *dblParameter*);

Parameter	Beschreibung
<i>bstrObjectId</i>	Ein String, der die ID des Objekts enthält.
<i>dblParameter</i>	Ein Feld mit drei Werten. Hier wird die neue Position der Drehung (Rotation) zum Ursprung angegeben.

x	0
y	1
z	2

**Rückgabewert**

keiner

**Beispiel****Beispiel**

```
sub main (i d)
  Dim angle
  angle = Graphi c. GetRotati on(i d)
  x=angl e(0)
  y=angl e(1)
  z=angl e(2)

  angl e(0)=1. 5
  angl e(1)=1. 5
  angl e(2)=1. 5
  call Graphi c. SetRotati on(i d, angl e)
end sub
```

## 3.5. Klasse ScriptItemDialog

Methode	Beschreibung
<b>MessageBox</b>	Zeigt einen Standard-Mitteilungsdialog an.
<b>MessageBoxExt</b>	Zeigt einen Mitteilungsdialog an (wählt Symbol und Buttons, z. B. Abbruch/Wiederholen/Ignorieren).
<b>CreateInputControl</b>	Vordefinition eines Steuerelements für ein Eingabefeld (Textfeld).
<b>ModifyInputControl</b>	Verändert ein Bearbeitungssteuerelement für ein Eingabefeld (nur Lesen, Längen- und Wertprüfung).
<b>InputBox</b>	Zeigt eine Eingabeaufforderung in einem Dialogfeld an, wartet auf die Eingabe eines Textes, oder auf das Klicken auf eine Schaltfläche und gibt einen Wert vom Typ String zurück, der den Inhalt des Textfeldes angibt.
<b>GetInputControlValue</b>	Empfängt die gegenwärtigen Werte des Steuerelements.
<b>GetValueCB</b>	Gibt die gegenwärtigen Werte von OpenProject Typ des Controls "ComboBox". (Skriptaktion erforderlich)
<b>GetValueCKB</b>	Gibt die gegenwärtigen Werte von OpenProject Typ des Controls "CheckBox". (Skriptaktion SelectProject erforderlich)
<b>GetValueEdit</b>	Gibt die gegenwärtigen Werte von OpenProject Typ des Controls "Edit".
<b>SetValueCB</b>	Setzt die Werte von OpenProject Typ des Controls "ComboBox". (Skriptaktion erforderlich)
<b>SetValueCKB</b>	Setzt die Werte von OpenProject Typ des Controls "CheckBox". (Skriptaktion erforderlich)
<b>SetValueEdit</b>	Setzt die Werte von OpenProject Typ des Controls "Edit". (Skriptaktion erforderlich)
<b>FileSelector (PE 5.8)</b>	Zeigt ein Standard Dateiauswahlfenster an.
<b>FileSelector (PE 5.9)</b>	Zeigt ein Standard Dateiauswahlfenster an.
<b>PropSetValues</b>	Schreibt oder hängt Werte an ein Attribut (Skriptaktion abhängig).
<b>PropGetValues</b>	Liest die möglichen Werte eines Attributes in einem Eigenschaftsdialog (Skriptaktion abhängig).
<b>PropGetCurrentValue</b>	Liest die tatsächlichen Werte einer Combobox, Listbox oder Radiobuttons (Skriptaktion abhängig).
<b>PropModifyAccess</b>	Ändert die Zugriffsrechte eines Feldes. (Skriptaktion erforderlich)
<b>PropModifyBGColor</b>	Ändert die Hintergrundfarbe. (Skriptaktion erforderlich)
<b>PropModifyFGColor</b>	Ändert die Vordergrundfarbe. (Skriptaktion erforderlich)
<b>PropModifyFontSize</b>	Ändert die Schriftgröße. (Skriptaktion erforderlich)
<b>PropModifyFontStyle</b>	Ändert den Schriftstil. (Skriptaktion erforderlich)
<b>PropModifyFontType</b>	Ändert die Schriftart. (Skriptaktion erforderlich)
<b>PropModifyRep</b>	Ändert die Anzeige (Skriptaktion erforderlich)
<b>PropModifyStyle</b>	Ändert den Typ des Controls (Skriptaktion abhängig).
<b>PropModifyVisibility</b>	Ändert das Layout.
<b>Redraw</b>	Dialog neuer Entwurf (Skriptaktion abhängig).
<b>RedrawAttribute</b>	Neuer Entwurf für Attribut (Skriptaktion abhängig).
<b>RedrawGroup</b>	Neuer Entwurf für Gruppe Redraw group (Skriptaktion abhängig)
<b>RedrawPage</b>	Neuer Entwurf für Seite (Skriptaktion abhängig)
<b>ResetItem</b>	Zurücksetzen des Skriptes in seinen Anfangsstatus.

### 3.5.1. Liste aller XScriptItemDialog Methoden

#### 3.5.1.1. MessageBox

##### SYNTAX

MessageBox( bstrCaption, bstrText) ;

Parameter	Beschreibung
BstrCaption	Dialog Überschrift
BstrText	Angezeigter Text

##### Rückgabewert

kein Rückgabewert

##### Beispiel

##### Beispiel

```
REM Displays a standard message box
REM entry parameter [id] is not used in this skript
sub main(id)
    capti on = "Mei ne erste MessageBox"
    message = "Hal lo worl d! "
    cal l Di alog. MessageBox(capti on, message)
end sub
```

#### 3.5.1.2. MessageBoxExt

##### SYNTAX

MessageBoxExt(bstrCaption, bstrText, bstrType);

Parameter	Beschreibung
bstrCaption	Dialog Überschrift.
bstrText	Angezeigter Text.
bstrType	Typ der angezeigten Message Box (siehe Rückgabewerte)

##### Rückgabewert

int\* iRetVal (abhängig vom Wert des bstrType)

BstrType	Rückgabewert	Beschreibung
ABORT_RETRY_IGNORE	3	Abbrechen (ABORT)
	4	Wiederholen (RETRY)
	5	Ignorieren (IGNORE)
ALL_ONE_NO	6	Alle (ALL)
	10	Eines (ONE )
	7	Nein (NO)
ERROR	1	OK
OK_CANCEL	1	OK
	2	Abbruch (CANCEL)
WARNING	1	OK
YES_NO	6	Ja (YES )
	7	Nein (NO)
YES_NO_CANCEL	6	Ja
	7	Nein (NO)
	2	Abbruch (ABORT)



OK	1	OK
----	---	----

**Beispiel:**

### Beispiel

```
REM Zeigt unterschiedliche Typen von Meldfenstern
REM entry parameter [id] is not used in this skript
sub main (id)
  Di spl ayMessage("ABORT_RETRY_I GNORE")
  Di spl ayMessage("ALL_ONE_NO")
  Di spl ayMessage("ERROR")
  Di spl ayMessage("OK_CANCEL")
  Di spl ayMessage("WARNI NG")
  Di spl ayMessage("YES_NO")
  Di spl ayMessage("YES_NO_CANCEL")
  Di spl ayMessage("OK")
end sub
sub Di spl ayMessage(howtodi spl ay)
  RetVal =Di al og. MessageBoxExt("Capti on", "Message", howtodi spl ay)
  call Di al og. MessageBox("Resul t", "Return Value is: " & RetVal)
end sub
```

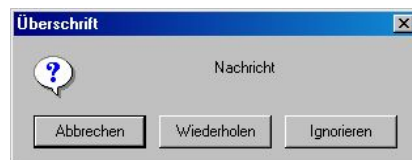


Abbildung 32: Beispiel: Dialog mit Buttons Abbrechen/Wiederholen/Ignorieren

### 3.5.1.3. CreateInputControl

#### SYNTAX

CreateInputControl ( bstrCtrlId, bstrControlType, bstrAttribute, vDefaultVal ue);

Parameter	Beschreibung
bstrCtrlId	Eine eindeutige String-Kennung für ein Steuerelement.
bstrControlType	Eine gültige Unterscheidung für den Typ des Steuerelements.(siehe auch Hinweis)
bstrAttribute	Zeigt die Eingabeaufforderung des Steuerelements an.
vDefaultValue	Zeigt den Default Wert des Steuerelements an oder eine Liste von Werten im Falle von "Optionsfeld" und "ComboBox".

#### Rückgabewert

kein Rückgabewert

#### Mögliche Werte von **bstrControlType**

EditString	Das Bearbeitungssteuerelement welches Strings enthält.
EditStringNoLowerCase	Bearbeitungssteuerelement, das Strings enthält, die keine Kleinbuchstaben enthalten.
EditInt	Bearbeitungssteuerelement, das Integer enthält
EditDouble	Bearbeitungssteuerelement, das Doubles enthält.
MultiLineEdit	Ein mehrzeiliges Textsteuerelement.
CheckBox	Ein Kontrollkästchen (Checkbox). Der Wert <b>0</b> bedeutet nicht gewählt, während <b>1</b> gewählt bedeutet.
RadioButtons	Ein „Optionsfeld“ Steuerelement. Eine Liste von Stichpunkten für die Radio Buttons muss weitergegeben werden. Die Buttons werden von 0 beginnend intern nummeriert.
ComboBox	Ein Kombinationsfeld-Steuerelement ( <b>ComboBox</b> ) kombiniert

die Merkmale eines Textfeld-Steuerelements und eines Listfeld-Steuerelements. Sie können entweder einen neuen Wert, wie in ein Textfeld-Steuerelement, eingeben, oder einen vorhandenen Wert, wie bei einem Listfeld-Steuerelement, auswählen. Es muss eine Liste von Werte Paaren weitergegeben werden (sichtbare Information und interne Information).

### Beispiel

#### Beispiel

```
' Erzeugt ein Dialog mit Steuerelementen
sub main(id)
    ' Define Edit Control for InputBox (Parameter: control id,
    control type, prompt, default value)
    call Dialog.CreateInputControl ("1", "EditString", "Ihr
    Name", "Peter")
    ...
    ' Define Combo Box Control (first value in array is default)
    Dim combo(1,1)
    combo(0,0) = "Sichtbarer Name 1"
    combo(0,1) = "Interer Name 1"
    combo(1,0) = "Visible Name 2"
    combo(1,1) = "Internal Name 2"
    call Dialog.CreateInputControl ("6", "ComboBox", "Combo Se-
    lection", 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", "Radi o
    Selection", radio)
    ' display InputBox caption "Ihre Daten"
    ret = Dialog.InputBox("Your Data")
    if ret=1 then
        ' check content of edit controls by means of control id (see
        above)
        name=Dialog.GetInputControl Value("1")
        comboval =Dialog.GetInputControl Value("6")
        radioval =Dialog.GetInputControl Value("7")
        message="Name: " & name
    else
        message="Di al og cancel ed. "
    end if
    ' Display message box
    call Dialog.MessageBox("Check Input", message)
end sub
```



**Hinweis:** Die beiden Buttons "Hilfe" (Help) und "Standard" (Default) haben keine Funktion und sind deshalb ausgegraut.



**Abbildung 33: Beispiel eines Dialoges**

**3.5.1.4. ModifyInputControl****SYNTAX**

ModifyInputControl (*bstrCtrlId*, *bstrCtrlTopic*, *vValue*);

Parameter	Beschreibung
<i>bstrCtrlId</i>	Eine eindeutige String-Kennung für das Steuerelement.
<i>bstrCtrlTopic</i>	Eine gültige Unterscheidung für den Inhalt, der gesteuert werden soll, z.B. "ReadOnly".
<i>vValue</i>	Eine gültiger Wert für unterschiedlichen Inhalt

**Mögliche Werte von *bstrCtrlTopic***

<i>ReadOnly</i>	Das Steuerelement wird schreibgeschützt angezeigt, wenn „true“ übergeben wird.	Gültig für alle Steuerelementtypen.
<i>MaxChar</i>	Die Länge der Eingabe ist durch MaxChar begrenzt. Der Standard ist Systemabhängig (32bit = 32767)	Gültig für alle Bearbeitungssteuerelemente
<i>MinLimit</i>	Der untere Grenzwert der Eingabe ist durch MinLimit begrenzt. Der Standard ist systemabhängig (32bit = 2147483648)	Gültig für alle Nummern der Bearbeitungssteuerelemente.
<i>MaxLimit</i>	Der obere Grenzwert der Eingabe ist durch MaxLimit begrenzt (32bit = 2147483647)	Gültig für alle Nummern Bearbeitungssteuerelemente.

**Rückgabewert**

kein Rückgabewert

**Beispiel****Beispiel**

'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", "ABCDEFGH")

call Dialog.ModifyInputControl ("3", "MaxChar", 20)

**end sub**

**3.5.1.5. InputBox****SYNTAX**

CreateInputControl (*bstrCaption*);

Parameter	Beschreibung
<i>bstrCaption</i>	Titel des Menüs ‚Dateiauswahl‘.

**Rückgabewert**

-1	Wenn das Objekt von dieser Klasse abgeleitet ist.
0	Wenn das Objekt nicht von dieser Klasse abgeleitet ist



**Hinweis:** In VBSkript wird -1 zu True oder vbTrue ausgewertet, in JScript wird +1 zu true ausgewertet.

**Beispiel****Beispiel**

```
' Erzeugt ein Dialog mit Steuerelementen
'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", "Pe-
ter")
...
' 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 abgebrochen. "
end if
' Display message box
call Dialog.MessageBox("Check Input", message)
end sub
```

### 3.5.1.6. GetInputControlValue

#### SYNTAX

GetInputControlValue(**bstrCtrlId**);

Parameter	Beschreibung
<b>BstrCtrlId</b>	Die String-ID des Steuerelements.

#### Rückgabewert

Aktueller Rückgabewert des Controls	Wenn das Dialogfenster mit "OK" verlassen worden ist.
Empty	Wenn das Dialogfenster mit "Abbruch" verlassen worden ist.

Mögliche Rückgabewerte abhängig vom **Controltyp**

<b>EditString</b>	Ein String.
<b>EditInt</b>	Ein Integer.
<b>EditDouble</b>	Ein Double Wert (Fließkommazahl).
<b>MultiLineEdit</b>	Ein String.
<b>CheckBox</b>	0 wenn gewählt, 1 wenn nicht gewählt.
<b>RadioButtons</b>	Die Sortiernummer des gewählten Radio Buttons (die Sortiernummer des ersten Buttons ist 0).
<b>ComboBox</b>	Der interne Wert der Combobox (nicht sichtbar).

#### Beispiel

#### Beispiel

```
'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", "Pe-
ter")
...
'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."
endif
'Display message box
call Dialog.MessageBox("Check Input", message)
end sub
```

### 3.5.1.7. GetValueCB

#### SYNTAX

GetValueCB (ctrlType; varCBVals);

Parameter	Beschreibung
ctrlType	Ein Identifikator für die Kontrolle.
varCBVals	Eine Liste mit den Werten der "ComboBox"

#### Rückgabewert

Wenn erfolgreich, dann erscheint eine Reihe von Comboboxeinträgen.

Mögliche Rückgabewerte für ctrlType

0	GLOBAL_FILTER
1	PRODUCT_FILTER
2	PROCESS_FILTER
3	RESOURCE_FILTER

### 3.5.1.8. GetValueCKB

#### SYNTAX

GetValueCKB (ctrlType);

Parameter	Beschreibung
ctrlType	Ein Identifikator für die Kontrolle.

#### Rückgabewert

Wenn erfolgreich, dann erscheint eine Reihe von Comboboxeinträgen.

### 3.5.1.9. GetValueEdit

#### SYNTAX

GetValueEdit (ctrlType);

Parameter	Beschreibung
ctrlType	Ein Identifikator für die Kontrolle.

#### Rückgabewert

Wenn erfolgreich, dann erscheint der Edit Control Eintrag.

**3.5.1.10. SetValueCB****SYNTAX**

SetValueCB (ctrlType; pvarValues);

Parameter	Beschreibung
ctrlType	Ein Identifikator für die Kontrolle.
pvarValues	Eine Liste mit den Werten der "ComboBox"

**Rückgabewert**

Keiner.

Mögliche Rückgabewerte für ctrlType

0	GLOBAL_FILTER
1	PRODUCT_FILTER
2	PROCESS_FILTER
3	RESOURCE_FILTER

**3.5.1.11. SetValueCKB****SYNTAX**

SetValueCKB (ctrlType; iCKBVal);

Parameter	Beschreibung
ctrlType	Ein Identifikator für die Kontrolle.
iCKBVal	"1" geprüft, "0" ungeprüft

**Rückgabewert**

Keiner

**3.5.1.12. SetValueEdit****SYNTAX**

SetValueEdit (ctrlType; bstrEditVal);

Parameter	Beschreibung
ctrlType	Ein Identifikator für die Kontrolle.
bstrEditVal	Ein Stringeintrag.

**Rückgabewert**

Keiner



**3.5.1.13. FileSelector (PE 5.7)****SYNTAX**

**FileSelector**(*bstrCaption*, *bstrDefaultFile*, *vFilterArray*);

Parameter	Beschreibung
<i>bstrCaption</i>	Auswahl des Dateiauswahlmenüs.
<i>bstrDefaultFile</i>	Eine gültige Default Datei mit Pfad Information. Es können Wildcards verwendet werden. Ist keine Datei angegeben, wird das System TEMP Verzeichnis gewählt.
<i>vFilterArray</i>	Ein Feld von Text / Erweiterungspaaren, die zum Filtern der angezeigten Dateien verwendet werden. (siehe Beispiel) Wenn ein Feld leer ist, ist das Filtern deaktiviert. Anderenfalls ist der erste Eintrag (Paar) Default.

**Rückgabewert**

Von *bstrSelectedFile*

Name (incl. Pfad) der selectierten Datei	Wenn erfolgreich und Öffnen (Open) ausgewählt wurde
Empty	anderenfalls

**Beispiel****Beispiel**

```

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, default_filename, filter)
    if filename <> "" then
        MsgBox(filename)
    else
        MsgBox("Keine Datei selektiert!")
    end if
end sub

```

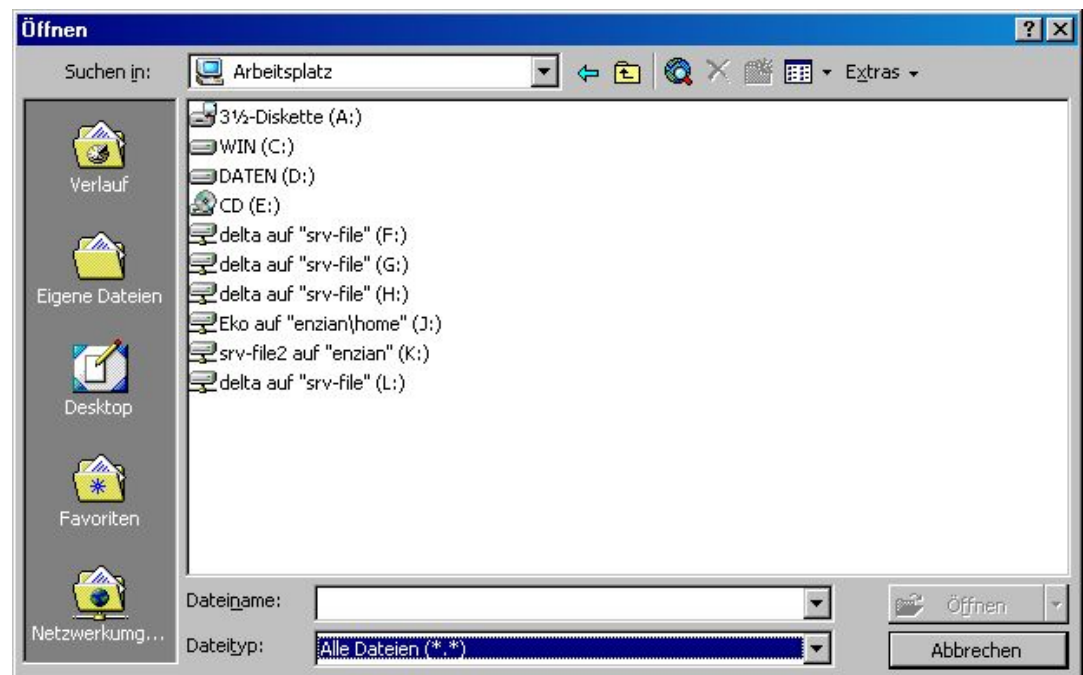


Abbildung 34: Dateiselektor PE 5.7

### 3.5.1.14. FileSelector (PE 5.9)

#### SYNTAX

**FileSelector**(*bstrMode*, *bstrCaption*, *bstrDefaultFile*, *vFilterArray*);

Parameter	Beschreibung
<i>bstrMode</i>	"Öffnen" oder "Speichern unter" hängt davon ab, wie der Dialogbutton angezeigt werden soll.
<i>bstrCaption</i>	Auswahl des Dateiauswahlmenüs.
<i>bstrDefaultFile</i>	Eine gültige Standard Datei mit Pfadinformation. Es können Wildcards verwendet werden. Ist keine Datei angegeben, wird das System TEMP Verzeichnis gewählt.
<i>vFilterArray</i>	Ein Feld von Text / Erweiterungspaaren, die zum Filtern der angezeigten Dateien verwendet werden. (siehe Beispiel) Wenn ein Feld leer ist, ist das Filtern deaktiviert. Anderenfalls ist der erste Eintrag (Paar) Standard.

#### Rückgabewert

Mögliche Rückgabewerte **von BSTR** *bstrSelectedFile*

<b>Name (incl. Pfad) der gewählten Datei</b>	Wenn eine Datei ausgewählt worden ist.
<b>Empty</b>	anderenfalls

#### Beispiel

#### Beispiel

```
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="Datei auswählen..."
    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, default_filename, filter)
if filename <> "" then
    MsgBox(filename)
else
    MsgBox("Keine Datei selektiert!")
end if
end sub

```

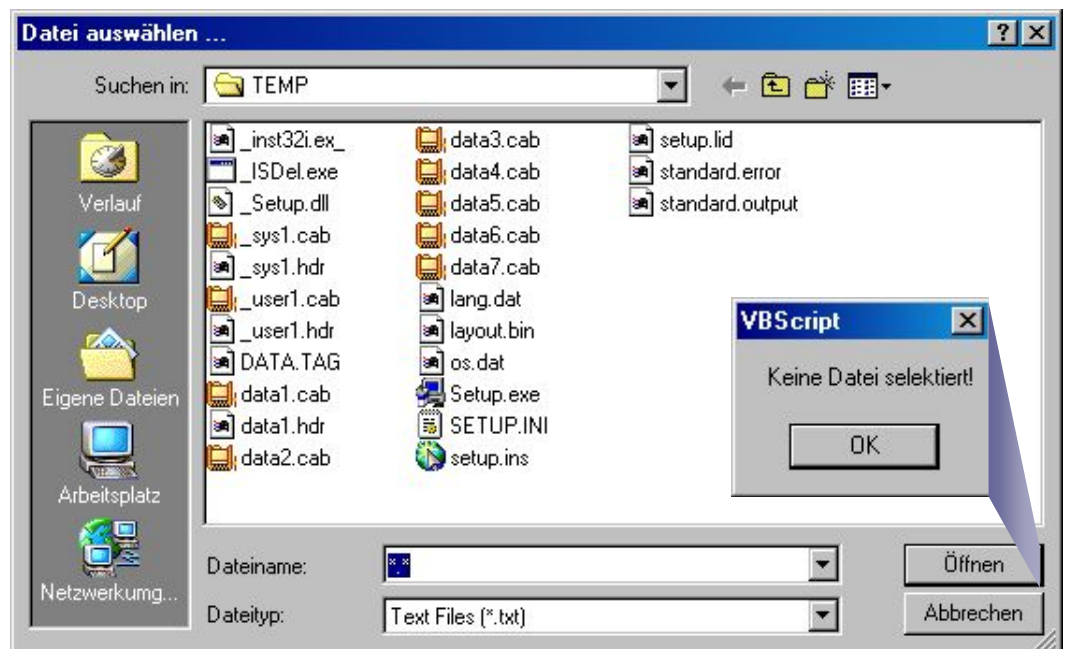


Abbildung 35: Dateiselektor PE 5.9

**3.5.1.15. PropSetValues****SYNTAX**

**PropSetValues**(*bstrObjectId*, *bstrAttributeId*, *iLinkType*, *vVarArray*);

Parameter	Beschreibung
<i>bstrObjectId</i>	Die ID des betrachteten Objektes
<i>bstrAttributeId</i>	Name des Attributs (= eindeutiger Schlüssel) dessen Typ des Controls bearbeitet werden soll.
<i>iLinkType</i>	<b>0 = Verknüpfen</b> (hängen die Werte des Skriptes an die Werte die aus der Datenbank gelesenen wurden) <b>1 = Ersetzen</b> (ersetzt die Originalwerte durch die Werte die im Skript erzeugt werden)
<i>vVarArray</i>	Das Feld (array) der neuen Werte. Die Dimensionierung des Feldes hängt von dem Typ des Controls ab. Für Comboboxen und Listboxen wird in der ersten Spalte des Feldes der <i>sichtbare</i> Wert, in der zweiten Spalte der <i>interne</i> Wert eingetragen.

**Rückgabewert**

keiner

**Beispiel****Beispiel**

```

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) ' vVarArray

  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 ' iLinkType

  call Dialog.PropSetValues(object_id, "carbodyposition",
  use_external, Values)

end if

...
end sub

```



**Hinweis:** Aufgrund des modalen Charakters der Eigenschaftendialoge kann diese Funktion nur zusammen mit den Skriptaktionen **Init Properties** und **Change Properties** benutzt werden.

### 3.5.1.16. PropGetValues

#### SYNTAX

PropGetValues(*bstrObjectId*, *bstrAttributeId*, *iLinkType*);

Parameter	Beschreibung
<i>bstrObjectId</i>	Die ID des betrachteten Objektes
<i>bstrAttributeId</i>	Name des Attributs (= eindeutiger Schlüssel) dessen Typ des Controls bearbeitet werden soll.
<i>iLinkType</i>	0 = liest alle Werte (Datenbankwerte und Werte die vom Skript erzeugt wurden). 1 = externe Werte (es werden nur die Werte gelesen, die von dem Skript erzeugt wurden).

#### Rückgabewert

Ein Feld von Werten. Die Größe und die Dimension des Feldes hängt von dem Typ des Controls ab. Ist der Typ des Controls z. B. *Edit*, besteht die Feldgröße nur aus einem einzelnen Wert. Bei Comboboxen und Listboxen ist der Rückgabewert ein zweidimensionales Feld.

#### Beispiel

#### Beispiel

```
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), "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
```



**Hinweis:** Aufgrund des modalen Charakters der Eigenschaftendialoge kann diese Funktion nur zusammen mit den Skriptaktionen *Init Properties* und *Change Properties* benutzt werden.

### 3.5.1.17. PropGetCurrentValue

#### SYNTAX

PropGetCurrentValue(*bstrObjectId*, *bstrAttributId*);

Parameter	Beschreibung
<i>bstrObjectId</i>	Die ID des betrachteten Objektes
<i>bstrAttributId</i>	Name des Attributs (= eindeutiger Schlüssel), dessen Typ des Controls bearbeitet werden soll.

#### Rückgabewert

Gibt den gegenwärtigen Inhalt eines Attributes zurück. Kann nur für Combo- und Listboxen sowie für Radiobuttons verwendet werden.

#### Beispiel

#### Beispiel

```
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 & ")"
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
```



**Hinweis:** Aufgrund des modalen Charakters der Eigenschaftendialoge kann diese Funktion nur zusammen mit den Skriptaktionen **Init Properties** und **Change Properties** benutzt werden.

**3.5.1.18. PropModifyAccess****SYNTAX**

```
PropModifyAccess(bstrObjectId,
                 layoutObjectType,
                 layoutObjectId,
                 iMode);
```

Parameter	Beschreibung
bstrObjectId	Die ID des betrachteten Objektes
layoutObjectType	Der Typ des betrachteten Objekts
bstrAttributeId	Name des Attributs (= eindeutiger Schlüssel), dessen Typ des Controls bearbeitet werden soll.
iMode	Zugangsmodus: "1" nur lesen und "0" beschreibbar

**Rückgabewert**

Keiner

**Beispiel****Beispiel**

```
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
```

**3.5.1.19. PropModifyBGColor****SYNTAX**

```
PropModifyBGColor(bstrObjectId,
                  layoutObjectType,
                  layoutObjectId,
                  bstrColor);
```

Parameter	Beschreibung
bstrObjectId	Die ID des betrachteten Objektes
layoutObjectType	Der Typ des betrachteten Objekts
bstrAttributeId	Name des Attributs (= eindeutiger Schlüssel), dessen Typ des Controls bearbeitet werden soll.
iControlBGColor	Farbe z.B: "#FFFFFF" (weiss), "#000000" (schwarz) etc.

**Rückgabewert**

Keiner

**Beispiel****Beispiel**

```
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
```

### 3.5.1.20. PropModifyFGColor

#### SYNTAX

```
PropModifyFGColor(bstrObjectId,  
                  layoutObjectType,  
                  layoutObjectId,  
                  bstrColor);
```

Parameter	Beschreibung
<i>bstrObjectId</i>	Die ID des betrachteten Objektes
<i>layoutObjectType</i>	Der Typ des betrachteten Objekts
<i>bstrAttributId</i>	Name des Attributs (= eindeutiger Schlüssel), dessen Typ des Controls bearbeitet werden soll.
<i>iControlFGColor</i>	Farbe z.B: "#FFFFFF" (weiss), "#000000" (schwarz) etc.

#### Rückgabewert

Keiner

#### Beispiel

#### Beispiel

```
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
```

### 3.5.1.21. PropModifyFontSize

#### SYNTAX

```
PropModifyFontSize(bstrObjectId,  
                   layoutObjectType,  
                   layoutObjectId,  
                   bstrSize);
```

Parameter	Beschreibung
<i>bstrObjectId</i>	Die ID des betrachteten Objektes
<i>layoutObjectType</i>	Der Typ des betrachteten Objekts
<i>bstrAttributId</i>	Name des Attributs (= eindeutiger Schlüssel), dessen Typ des Controls bearbeitet werden soll.
<i>iControlStyle</i>	Gegenwärtig werden nur die folgenden Werte unterstützt 0 (default) und 1 (Kontrolle wird auf "read only" gestellt).

#### Rückgabewert

Keiner

#### Beispiel

#### Beispiel

```
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
```



**3.5.1.22. PropModifyFontStyle****SYNTAX**

```
PropModifyFontStyle(bstrObjectId,  
                   layoutObjectType,  
                   layoutObjectId,  
                   bstrStyle);
```

Parameter	Beschreibung
<i>bstrObjectId</i>	Die ID des betrachteten Objektes
<i>layoutObjectType</i>	Der Typ des betrachteten Objekts
<i>bstrAttributId</i>	Name des Attributs (= eindeutiger Schlüssel), dessen Typ des Controls bearbeitet werden soll.
<i>bstrStyle</i>	Schriftstil: "B" Fett, "I" Kursiv, "U" unterstrichen oder Kombinationen z.B.: "B U I".

**Rückgabewert**

Keiner

**Beispiel****Beispiel**

```
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
```

**3.5.1.23. PropModifyFontType****SYNTAX**

```
PropModifyFontType(bstrObjectId,  
                  layoutObjectType,  
                  layoutObjectId,  
                  bstrStyle);
```

Parameter	Beschreibung
<i>bstrObjectId</i>	Die ID des betrachteten Objektes
<i>layoutObjectType</i>	Der Typ des betrachteten Objekts
<i>bstrAttributId</i>	Name des Attributs (= eindeutiger Schlüssel), dessen Typ des Controls bearbeitet werden soll.
<i>bstrType</i>	Schriftart z.B.: "MS Sans Serif" etc.

**Rückgabewert**

Keiner

**Beispiel****Beispiel**

```
function sa_i ni tproperties(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
```

**3.5.1.24. PropModifyRep****SYNTAX**

```
PropModifyRep(bstrObjectId,
              layoutObjectType,
              layoutObjectId,
              bstrStyle);
```

Parameter	Beschreibung
<code>bstrObjectId</code>	Die ID des betrachteten Objektes
<code>layoutObjectType</code>	Der Typ des betrachteten Objekts
<code>bstrAttributeId</code>	Name des Attributs (= eindeutiger Schlüssel), dessen Typ des Controls bearbeitet werden soll.
<code>bstrStyle</code>	Repräsentiert eine Kombination von Farbe und Schriftänderungen auf einmal: z.B.: "<bgcolor>#F00000</bgcolor><fgcolor>FF0000<fontsize>20</fontsize><fontstyle>U</fontstyle><fonttype>Arial</fonttype>"

**Rückgabewert**

Keiner

**Beispiel****Beispiel**

```
function sa_i ni tproperties(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
```

**3.5.1.25. PropModifyStyle****SYNTAX**

```
PropModifyStyle(bstrObjectId,  
               bstrAttributeId,  
               iControlStyle);
```

Parameter	Beschreibung
<i>bstrObjectId</i>	Die ID des betrachteten Objektes
<i>bstrAttributeId</i>	Name des Attributs (= eindeutiger Schlüssel), dessen Typ des Controls bearbeitet werden soll.
<i>iControlStyle</i>	Gegenwärtig werden nur die folgenden Werte unterstützt 0 (default) und 1 (Kontrolle wird auf "read only" gestellt).

**Rückgabewert**

keiner

**Beispiel****Beispiel**

```
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
```



**Hinweis:** Aufgrund des modalen Charakters der Eigenschaftendialoge kann diese Funktion nur zusammen mit den Skriptaktionen [Init Properties](#) und [Change Properties](#) benutzt werden.

**3.5.1.26. PropModifyVisibility****SYNTAX**

```
PropModifyStyle(bstrObjectId,  
               layoutObjectType,  
               layoutObjectId,  
               iMode);
```

Parameter	Beschreibung
<i>bstrObjectId</i>	Die ID des betrachteten Objektes
<i>layoutObjectType</i>	Der Typ des betrachteten Objekts
<i>bstrAttributeId</i>	Name des Attributs (= eindeutiger Schlüssel), dessen Typ des Controls bearbeitet werden soll.
<i>iMode</i>	Sichtbarkeitsmodus: "0" zeigen und "1" verstecken.

**Rückgabewert**

keiner

**Beispiel****Beispiel**

```
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
```

**3.5.1.27. Redraw****SYNTAX**

Redraw(*bstrObjectId*);

Parameter	Beschreibung
<i>bstrObjectId</i>	Die ID des betrachteten Objektes

**Rückgabewert**

keiner

**Beispiel****Beispiel**

```
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
```

**3.5.1.28. RedrawAttribute****SYNTAX**

RedrawAttribute(*bstrObjectId*;  
*attributeName*);

Parameter	Beschreibung
<i>bstrObjectId</i>	Die ID des betrachteten Objektes
<i>attributeName</i>	Name des Attributs (= eindeutiger Schlüssel), dessen Typ des Controls bearbeitet werden soll.

**Rückgabewert**

keiner

**Beispiel****Beispiel**

```
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
```

**3.5.1.29. RedrawGroup****SYNTAX**

```
RedrawGroup(bstrObjectld;
            groupld);
```

Parameter	Beschreibung
bstrObjectld	Die ID des betrachteten Objektes
groupld	Name der Gruppe (= eindeutiger Schlüssel), dessen Typ des Controls bearbeitet werden soll.

**Rückgabewert**

keiner

**Beispiel****Beispiel**

```
function sa_initproperties(object_ids)
oid = data.GetAttributebyld (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
```

**3.5.1.30. RedrawPage****SYNTAX**

```
RedrawPage(bstrObjectld;
            pageld);
```

Parameter	Beschreibung
bstrObjectld	Die ID des betrachteten Objektes
pageld	Nummer (= eindeutiger Schlüssel) von der Seite.

**Rückgabewert**

keiner

**Beispiel****Beispiel**

```
function sa_initproperties(object_ids)
oid = data.GetAttributebyld (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
```

## 3.6. Klasse ScriptItemList

Methode	Beschreibung
<b>CreateListView (bis PE 5.8)</b>	Erstellt eine Browser Listensicht, die die durchlaufenen Objekte enthält. Die Liste dient als Drag & Drop Quelle.
<b>CreateListView ab (PE 5.9)</b>	Erstellt eine Browser Listensicht, die die durchlaufenen Objekte enthält. Die Liste dient als Drag & Drop Quelle.
<b>OpenFinder</b>	Öffnet den Projektsucher mit vordefinierten Eigenschaften.
<b>ResetItem</b>	Zurücksetzen des Skriptes in seinen Anfangsstatus.

### 3.6.1. Liste aller XscriptItemList Methoden

#### 3.6.1.1. CreateListView(PE 5.8)

##### SYNTAX

```
CreateLi stVi ew(bstrParentObj ectId,
                bstrEmpty1,
                bstrEmpty2,
                vChi l dArray);
```

Parameter	Beschreibung
<b>bstrParentObjectld</b>	Ein String, der die ID des Vaterobjekts enthält.
<b>bstrEmpty1</b>	Wird nicht verwendet.
<b>bstrEmpty2</b>	Wird nicht verwendet.
<b>vChildArray</b>	Ein Feld, das die ID des Objekts enthält, das in der Liste angezeigt werden soll.

##### Rückgabewert

kein Rückgabewert

##### Beispiel (Auszug)

#### Beispiel

```
' 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, "relationobject2")
        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.CreateLi stVi ew(id, "List of chil dren", child_ids)
    end if
end sub
```

**3.6.1.2. CreateListView(PE 5.9)****SYNTAX**

```
CreateLi stVi ew( bstrParentObj ectId,
                  bstrCapti on,
                  vChi l dArray);
```

Parameter	Beschreibung
bstrParentObjectld	Ein String, der die ID des Vaterobjekts enthält.
bstrCaption	Die Auswahl des List view Fensters
vChildArray	Ein Feld, das die Objekt ID des Objekts enthält, das in der Liste angezeigt werden soll.

**Rückgabewert**

kein Rückgabewert

**Beispiel****Beispiel**

```
' 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 mai n(i d)
    parent_base_id = GetBase(i d)
    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, "ergocom
        base")
        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.CreateLi stVi ew(i d, "Test", "Test",
child_ids)
        ' R8 function call
        call List.CreateLi stVi ew(i d, "Li st of children",
child_ids)
    end if
end sub
function GetBase(i d)
    GetBase = Data.GetAttributebyId(i d, "rel ati onobject2")
end function
```

### 3.6.1.3. OpenFinder

#### SYNTAX

OpenFinder(*bstrProjectId*, *bstrTypeName*, *vSearchCriteria*);

Parameter	Beschreibung
<i>bstrProjectId</i>	Die ID des Projektes, für das der Sucher geöffnet werden soll.
<i>bstrTypeName</i>	Der Name des zu selektierenden Typen.
<i>vSearchCriteria</i>	Ein Feld, das zusätzliche Suchkriterien enthält.

#### Rückgabewert

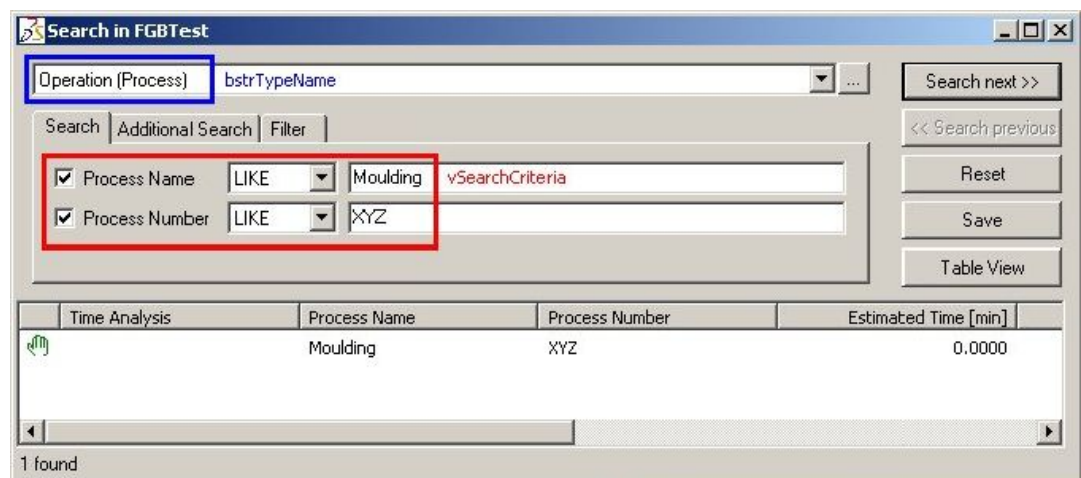
kein Rückgabewert

#### Beispiel

#### Beispiel

```
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
```



**Hinweis:** Das Öffnen des Projektsuchers mit vordefinierten Eigenschaften steigert nicht die Sucherleistung.



### 3.7. Klasse ScriptItemQuery

Suche auf der Datenbank	
Methode	Beschreibung
<b>SetQuery</b>	Mit <b>SetQuery</b> wird auf einer <i>Tabelle</i> (z. B. "ergocompproductdefault") im Attribut <i>Name</i> (z. B. "name") nach dem <i>Ausdruck</i> ("*BG*") und auf <i>Gleichheit</i> mit dem Operator (z.B. "=") geprüft. Die Gleichheitsoperatoren wurden der Konsistenz wegen auf (die vermutlich am häufigsten eingesetzte) VB-Syntax gemappt, also (=, <=, >=, <>).
<b>SetConcatenator</b>	Will man mehrere Suchvorgänge kombinieren, ist dies mittels <b>SetConcatenator</b> möglich. Damit werden mehrere Suchvorgänge aneinandergehängt. Zulässige Ausdrücke sind hier <b>"AND"</b> und <b>"OR"</b> . Der Tabellename dieser verknüpften Suchvorgänge muss identisch sein!
<b>GetFirstResult /GetNextResult</b>	Mit <b>GetFirstResult/GetNextResult</b> werden die Ergebnisse abgerufen. Danach (oder generell davor) muss die Suche mit <b>ResetSearch</b> abgeschlossen werden!
<b>GetNextResult</b>	Mit <b>GetNextResult</b> werden die nachfolgenden Ergebnisse abgerufen.
<b>GetResultCount</b>	Empfängt die Anzahl der Ergebnisse einer Abfrage.
<b>GetOnlyFirstResult</b>	Führt eine Abfrage aus und empfängt nur das erste Ergebnis
<b>ResetSearch</b>	Setzt die gegenwärtigen Abfragen zurück (wenn mehr als eine Abfrage in einem Skript verwendet wird).
<b>SetOrderAttribute</b>	Sortieren der Ergebnisse einer Abfrage nach einem vordefinierten Attribut.
<b>BeginSubQuery</b>	Definition des Beginns einer Unterabfrage.
<b>EndSubQuery</b>	Definition des Endes einer Unterabfrage.
<b>SetSubQuery</b>	Definition einer Unterabfrage.
<b>SetQueryMode</b>	Setzt den Modus einer Abfrage (nur anwendbar, wenn Unterabfragen benutzt werden).
<b>CacheResultIds</b>	Definiert, ob Ergebnisse von Abfragen gecacht werden sollen oder nicht.
<b>SetFetchingSize</b>	Wählt die Anzahl der Ergebnisse aus, die mit einem Serverzugriff geholt werden sollen.
<b>SetOQLQuery</b>	Erzeugt eine OQL Abfrage.
<b>IsObjectValid</b>	Überprüft, ob ein Objekt in Bezug auf die gegenwärtigen Filtereinstellungen gültig ist.
<b>UseProjectFilter</b>	Ermöglicht die Verwendung von Projektfiltren für die anschließende Query.
<b>IsFilterActive</b>	Überprüft, ob ein Filter gesetzt ist.
<b>ResetItem</b>	Zurücksetzen des Skriptes in seinen Anfangsstatus.

### 3.7.1. Liste aller ScriptItemQuery Methoden

#### 3.7.1.1. SetQuery

##### SYNTAX

SetQuery( *bstrSearchTable*, *bstrName*, *bstrOperator*,  
*varCompareTo*);

Parameter	Beschreibung
<i>bstrSearchTable</i>	Die Tabelle (das Objekt), auf der die Abfrage ausgeführt werden soll (z. B. "ergocomplantdefault").
<i>bstrName</i>	Das Attribut, auf dem die Bedingung angewendet werden soll
<i>bstrOperator</i>	Der Vergleichsoperator (z. B. "=" (gleich)).
<i>varCompareTo</i>	Der Wert, mit dem der gängige Wert von <i>bstrName</i> verglichen werden soll.

##### Mögliche Werte von *bstrOperator*

"="	Wert von <i>bstrName</i> gleich der zu vergleichenden <i>varCompare</i> .
"<>"	Wert von <i>bstrName</i> ungleich <i>varCompareTo</i> .
"<"	Wert von <i>bstrName</i> ist kleiner als <i>varCompareTo</i> .
"<="	Wert von <i>bstrName</i> ist kleiner gleich als <i>varCompareTo</i> .
">"	Wert von <i>bstrName</i> ist größer als <i>varCompareTo</i> .
">="	Wert von <i>bstrName</i> ist größer oder gleich als <i>varCompareTo</i> .
" CS"	Case Sensitive (berücksichtigung der Groß-Klein Schreibung) Abfrage - Bei Suchen nach Zeichenketten wird „ CS“ nach dem Vergleichsoperator eingetragen, z.B. "<= CS"; "= CS"; ">= CS"; etc.
"NO"	Achtet auf kein Attribut (gibt alle Objekte aus <i>bstrSearch</i> Tabelle zurück).

Folgende Erweiterungen sind möglich, wenn Sie nach benutzerdefinierten Attributen und/oder Datumswerten suchen

Erweiterung	Beschreibung	Beispiel
" AVSET"	Das Attribut ist benutzerdefiniert.	"myattr1 AVSET"
" AVSETDATE"	Das Attribut ist benutzerdefiniert und enthält einen Datumswert. Der Zeitan- teil des Datums wird ignoriert.	"myattr1 AVSETDATE"
" AVSETDATETIME"	Das Attribut ist benutzerdefiniert und enthält einen Datumswert.	"myattr1 AVSETDATETIME"
" DATE"	Das Attribut enthält einen Datumswert. Der Zeitteil des Datums wird ignoriert.	"myattr1 DATE"
" DATETIME"	Das Attribut enthält einen Datumswert.	"myattr1 DATETIME"

##### Rückgabewert

kein Rückgabewert

## Beispiel

## Beispiel

```

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.GetInputControl Value("1")
    attribute=Dialog.GetInputControl Value("2")
    operator=Dialog.GetInputControl Value("3")
    vartocompare=Dialog.GetInputControl Value("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 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, e.g. "ergoproject".

---

► Die Erweiterung für Case Sensitivity Abfragen ("CS") und für benutzerdefinierte Attribute sowie Datumsattribute ist ab der Version PE5.13SP3 verfügbar.

---

**3.7.1.2. SetConcatenator****SYNTAX**

SetConcatenator(*bstrConcatenator*);

Parameter	Beschreibung
<i>BstrConcatenator</i>	Ein String, der die Verkettung der nachfolgenden Abfragen beschreibt.

**Rückgabewert**

kein Rückgabewert

**Mögliche Werte von *bstrConcatenator***

"AND"	Objekte einer Abfrage, für die die Bedingung A und B wahr sind.
"OR"	Objekte einer Abfrage, für die die Bedingung A oder B wahr sind.
"("	Öffnende Klammer. Für verschachtelte Bedingungen.
")"	Schließende Klammer. Für verschachtelte Bedingungen.
"XOR"	Exclusives Oder. Nur gültig, wenn der Abfragemodus SetQueryMode = "CompareSQ".
"IN_PRECEDING_QUERY"	In den nachfolgenden Bedingungen werden nur die Ergebnisse einer vorhergehenden Abfrage verwendet. Nur gültig, wenn der Abfragemodus SetQueryMode ist "CompareSQ".

**Beispiel****Beispiel**

```
REM Advanced Query demonstrating usage of SetConcatenator
sub main (project_id)
    typearea="ergocompl antdefault"
    ...
    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
```

**3.7.1.3. GetFirstResult****SYNTAX**

GetFirstResult();

**Rückgabewert**

Die Objekt ID des ersten Ergebnisses oder ein Leerstring.

**Beispiel****Beispiel**

```
REM Wenn sie die Vorgaben benutzen, werden Sie alle Produktkom-
REM Das Skript muss dann auf dem aktuellen Projektknoten ausge-
REM führt werden, da die weitgereichte ID in der SetQuery Anwei-
REM sung benutzt wird
sub main (project_id)
def_tablename= "ergocomproductdefault"
def_attribute= "ergoproject"
def_operator= "="
def_vartocompare = project_id
call Dialog.CreateInputControl ("1", "EditString", "Objects (ta-
blename) 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, "=", 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
```

### 3.7.1.4. **GetNextResult**

#### **SYNTAX**

**GetNextResult** ();

#### **Rückgabewert**

Die Objekt ID des nächsten Ergebnisses oder ein Leerstring.

#### **Beispiel**

#### **Beispiel**

REM Wenn sie die Vorgaben benutzen, werden Sie alle Produktkomponenten des aktuellen Projekts finden.

REM Das Skript muss dann auf dem aktuellen Projektknoten ausgeführt werden, da die weitgereichte ID in der SetQuery Anweisung benutzt wird.

```
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.GetInputControl Value("1")
    attribute=Dialog.GetInputControl Value("2")
    operator=Dialog.GetInputControl Value("3")
    vartocompare=Dialog.GetInputControl Value("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
```



#### **Hinweis:**

Die Funktion muss **nach** dem Aufruf der Funktion **GetFirstResult** aufgerufen werden.

**3.7.1.5. GetResultCount****SYNTAX**

GetResultCount();

**Rückgabewert**

Die Nummer des Ergebnisses oder Null, wenn nichts in der Liste enthalten ist.

**Beispiel****Beispiel**

REM Wenn sie die Vorgaben benutzen, werden Sie alle Produktkomponenten des aktuellen Projekts finden.  
REM Das Skript muss dann auf dem aktuellen Projektknoten ausgeführt werden, da die weitergereichte ID in der SetQuery Anweisung benutzt wird

```
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, "=", 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
```



**3.7.1.6. GetOnlyFirstResult****SYNTAX**

GetOnlyFirstResult();

**Rückgabewert**

Die Objekt ID des ersten Ergebnisses oder ein Leerstring.

**Beispiel****Beispiel**

```
sub main(i d)
...
result_id=Query.GetOnlyFirstResult
end sub
```

**Hinweis:**

Ähnlich wie [GetFirstResult](#). Es wird aber nur ein Rückgabewert geliefert.

Ein Aufruf der Methode [GetNextResult](#) ist nicht möglich.

---

**3.7.1.7. ResetSearch****SYNTAX**

ResetSearch();

**Rückgabewert**

kein Rückgabewert

**Beispiel****Beispiel**

```
sub main(i d)
...
call Query.ResetSearch
end sub
```

**Hinweis:**

Alle vorausgegangenen **SetQuery**s und **SetSubQuery**s werden zurückgesetzt.

---

### 3.7.1.8. SetOrderAttribute

#### SYNTAX

SetOrderAttribute( *bstrSearchTable*, *bstrAttributeName*,  
*bstrOrder*);

Parameter	Beschreibung
<i>bstrSearchTable</i>	Die Tabelle (das Objekt), auf der die Abfrage ausgeführt werden soll (z. B. "ergocomplantdefault").
<i>bstrAttributeName</i>	Das Attribut, nach dem die Ergebnisse geordnet werden sollen.
<i>bstrOrder</i>	Aufsteigende oder absteigende Reihenfolge.

Mögliche Werte von *bstrOrder*

"ASC"	Sortiert aufsteigend
"DESC"	Sortiert absteigend

#### Rückgabewert

kein Rückgabewert

#### Beispiel

#### Beispiel

```
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
```

### 3.7.1.9. BeginSubQuery

#### SYNTAX

BeginSubQuery(*bstrQueryId*);

Parameter	Beschreibung
<i>bstrQueryId</i>	Jeder benutzerdefinierte String, der die Unterabfrage kennzeichnet.

#### Rückgabewert

kein Rückgabewert

#### Beispiel

#### Beispiel

```
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", "ergoproject",
        "=", "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
```

### 3.7.1.10. EndSubQuery

#### SYNTAX

EndSubQuery(*bstrQueryId*);

Parameter	Beschreibung
<i>bstrQueryId</i>	Jeder benutzerdefinierte String, der die Unterabfrage kennzeichnet.

#### Rückgabewert

Kein Rückgabewert

#### Beispiel

#### Beispiel

```
REM Benutzen einer Unterabfrage
REM 1. Findet alle Prozesskomponenten im Projekt "project_name"
REM Das Skript ist unabhängig von der Eingangs-ID
sub main (id)
    project_name="Project1"
    search_name="*Welt*"

    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
```

**3.7.1.11. SetSubQuery****SYNTAX**

```
SetSubQuery( bstrSearchTable, bstrName, bstrOperator,
bstrQueryId);
```

Parameter	Beschreibung
<code>bstrSearchTable</code>	Die Tabelle (das Objekt), auf der die Abfrage ausgeführt werden soll (z. B. "ergocomplantdefault").
<code>bstrName</code>	Das Attribut, auf das die Bedingung angewendet werden soll
<code>bstrOperator</code>	Der Vergleichsoperator (z. B. "=" (gleich)).
<code>varCompareTo</code>	Der Wert, mit dem der gegenwärtige Wert von <code>bstrName</code> verglichen werden soll.

**Mögliche Werte von *bstrOperator***

"="	Gegenwärtiger Wert von <code>bstrName</code> gleich <code>varCompareTo</code> .
"<>"	Gegenwärtiger Wert von <code>bstrName</code> ungleich <code>varCompareTo</code> .
"<"	Gegenwärtiger Wert von <code>bstrName</code> kleiner als <code>varCompareTo</code> .
"<="	Gegenwärtiger Wert von <code>bstrName</code> kleiner gleich als <code>varCompareTo</code> .
">"	Gegenwärtiger Wert von <code>bstrName</code> größer als <code>varCompareTo</code> .
">="	Gegenwärtiger Wert von <code>bstrName</code> größer gleich als <code>varCompareTo</code> .
"NO"	Achtet auf kein Attribut (gibt alle Objekte aus <code>bstrSearch Tabelle</code> zurück).

**Rückgabewert**

kein Rückgabewert

**Beispiel****Beispiel**

```
REM Das Skript ist unabhängig von der Eingangs 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("ergocomprocessdefault", "ergoproject", "=",
"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
```

### 3.7.1.12. SetQueryMode

#### SYNTAX

SetQueryMode ([bstrSearchMode](#));

Parameter	Beschreibung
<a href="#">bstrSearchMode</a>	Ein String, der den Suchmodus definiert. Default ist "Normal".

Mögliche Werte von **bstrSearchMode**

"Normal"	Normaler Abfragemodus (standard). Benutzbar in normalen Abfragen und wenn normale Abfragen mit Unterabfragen verglichen werden.
"CompareSQ"	Wenn Ergebnisse unterschiedlicher Unterabfragen verglichen werden sollen.

#### Rückgabewert

kein Rückgabewert

#### Beispiel

#### Beispiel

```
sub main(i d)
...
call Query.SetQueryMode("CompareSQ")
...
end sub
```

**3.7.1.13. CacheResultIds****SYNTAX**

CacheResultIds(bFlag);

Parameter	Beschreibung
BFlag	Ein globaler Flag, der dem Host anweist die abgerufenen Ergebnisse in einer Liste zu cachem, oder nicht zu cachem. Standard ist True.

**Rückgabewert**

kein Rückgabewert

**Beispiel****Beispiel**

```
sub main(id)
...
call Query.CacheResultIds(False)
...
end sub
```

**3.7.1.14. SetFetchingSize****SYNTAX**

SetFetchingSize(ulFetchingSize);

Parameter	Beschreibung
ulFetchingSize	Die Anzahl der Ergebnisse, die mit einem Serverzugriff geholt werden sollen. Standard ist 25.

**Rückgabewert**

kein Rückgabewert

**Beispiel****Beispiel**

```
sub main(id)
...
call Query.SetFetchingSize(10)
...
end sub
```

**3.7.1.15. SetOQLQuery****SYNTAX**SetOQLQuery(*bstrOQLQuery*);

Parameter	Beschreibung
<i>bstrOQLQuery</i>	Eine OQL Abfrage als Textstring.

**Rückgabewert**

kein Rückgabewert

**Beispiel****Beispiel**

```

sub main(i_d_notused)
REM OQL
Call Query.SetQueryMode("OQL")
oql query="SELECT * FROM XD0ErgoPI anTypeExtent type;"
Call Query.SetOQLQuery(oql query)
REM "Normal" Query
REM call Query.SetQuery("XD0ErgoPI anType", "", "NO", "")
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

```



**Hinweis:** Vor dem Anrufen von SetOQLQuery muss der Fragemodus auf "OQL" mit Hilfe von SetQueryMode gestellt werden.

**3.7.1.16. IsObjectValid****SYNTAX**IsObjectValid(*bstrObjectId*);

Parameter	Beschreibung
<i>bstrObjectId</i>	Die Objekt ID

**Beispiel****Beispiel**

```

sub main (project_id)
...
compval id = Query.IsObjectValid(result_id)
...
end sub

```



### 3.7.1.17. UseProjectFilter

#### SYNTAX

UseProjectFilter(*bstrProjectId*);

Parameter	Beschreibung
<i>bstrProjectId</i>	Die Projekt ID.

Sie müssen diese Funktion vor dem Filtern (GetFirstResult oder GetResultCount) aufrufen.

#### Rückgabewert

keiner

#### Beispiel

#### Beispiel

```
sub main (project_id)
...
if radioval = 1 then
all Query.UseProjectFilter(project_id)
end if
...
end sub
```

### 3.7.1.18. IsFilterActive

#### SYNTAX

IsFilterActive(*bstrProjectId*);

Parameter	Beschreibung
<i>bstrProjectId</i>	Die Projekt ID.

#### Rückgabewert

<i>bFlag</i>	<ul style="list-style-type: none"><li>• -1 (True) wenn ein Filter aktiviert ist.</li><li>• 0 (False) wenn kein Filter aktiviert ist.</li></ul>
--------------	--

#### Beispiel

#### Beispiel

```
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="ergocompl antdefault"
```

```
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
```

## 3.8. Klasse ScriptItemConfig

Methode	Beschreibung
<b>GetAllAttributes</b>	Gibt alle Attribute eines Typs oder Planungstyps zurück.
<b>GetAllPlanTypes</b>	Gibt alle Planungstypen eines bestimmten Typs zurück.
<b>GetAllTypes</b>	Gibt alle Konfigurationstypen zurück (incl. Plantypen).
<b>GetTypeInfoTN</b>	Get type info by (unique) type name.
<b>GetTypeInfo</b>	Gibt die Typ Information anhand des (eindeutigen) Typnamens zurück.
<b>GetAttributeInfoTN</b>	Gibt, anhand des Typnamens und des Attributnamens, Informationen zu einem Attribut.
<b>GetAttributeInfo</b>	Gibt, anhand der Attribut-ID, Informationen zu einem Attribut.
<b>GetAttributeValueList</b>	Gibt die Werteliste eines Attributes zurück.
<b>GetAttributeValueInfo</b>	Get attribute value info by attribute value ID.
<b>GetParentPCRelations</b>	Gibt alle parent child (PC) Relationen für einen bestimmten Eltern-Typ (oder Planungstyp) zurück.
<b>GetPCRelations</b>	Gibt alle Eltern-Kind-Beziehungen zwischen den durchlaufenen Eltern- und Kindertypen zurück.
<b>GetPCRelationInfo</b>	Gibt die PC Relation-Info anhand der ID der PC Relation zurück.
<b>ResetItem</b>	Zurücksetzen des Skriptes in seinen Anfangsstatus.

### 3.8.1. Liste aller XScriptItemConfig Methoden

#### 3.8.1.1. **GetAllAttributes**

##### SYNTAX

GetAllAttributes(**bstrTypeName**);

Parameter	Beschreibung
<b>bstrTypeName</b>	Eine Zeichenfolge, die den Namen des konfigurierten Typen oder des Planungstypen enthält (GUID).

##### Rückgabewert

Ein Feld, welches die eindeutigen IDs aller Attribute enthält.

##### Beispiel

#### Beispiel

```
sub main(id)
...
attr = Config.GetAllAttributes("script")
...
end sub
```

**3.8.1.2. GetAllPlanTypes****SYNTAX**

`GetAllPlanTypes(bstrProjectId, bstrPlanTypeSetId, bstrTypeId);`

Parameter	Beschreibung
<code>bstrProjectId</code>	Ein String, der die ID des Projektes enthält.
<code>bstrPlanTypeSetId</code>	Ein String, der die ID eines Planungstypensatzes enthält.
<code>bstrTypeId</code>	Ein String, der die ID eines Typen enthält.

**Rückgabewert**

Ein Feld, welches die eindeutigen IDs des Planungstypen in Abhängigkeit von den Eingangsparametern enthält.

Sie können entweder die Projekt-ID (jedes Projekt hat einen eindeutigen Planungstypensatz (PTS)) oder die PTS-ID selbst verwenden.

Sie können wahlweise die ID eines Typen verwenden. Wenn ein Typ verwendet wird, werden nur die Planungstypen, die dieser Typ besitzt, zurückgegeben.

**Beispiel****Beispiel**

```
sub main(project_id)
    plantypes = Config.GetAllPlanTypes(project_id, "", "ergocomp-plantdefault")
    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
```

**3.8.1.3. GetAllTypes****SYNTAX**

`GetAllTypes();`  
Keine Parameter

**Rückgabewert**

Ein Feld, welches die IDs aller Typen und Planungstypen der Datenbank enthält.

**Beispiel****Beispiel**

```
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
```



**Hinweis:** Für eine PTS abhängige Liste von Planungstypen siehe [GetAllPlanTypes](#).

### 3.8.1.4. GetTypeInfoTN

#### SYNTAX

GetTypeInfoTN (bstrTypeName, bstrProperty);

Parameter	Beschreibung
bstrTypeName	Eine Zeichenfolge, die den eindeutigen Namen eines Typen oder Planungstypen enthält.
bstrProperty	Eine Zeichenfolge, die den Namen der Eigenschaft enthält.

#### Rückgabewert

Der gegenwärtige Wert der geforderten Eigenschaft.

#### Beispiel

### Beispiel

```
sub main(project_id)
    uniquetypename = "ergocompproductdefault"
    property = "captionsingular"
    value = Config.GetTypeInfoTN(uniquetypename, property)
    MsgBox(value)
end sub
```



**Hinweis** : Ein eindeutiger Name eines Typs ist z. B. "ergocompproductdefault" oder "script".

Ein eindeutiger Name eines Planungstypen ist die GUID (Global Unique Identifier), z. B. "587f7aaf-5de0-417b-a6c9-64537fa31fc3" für einen Prozess.

### 3.8.1.5. GetTypeInfo

#### SYNTAX

GetTypeInfo *bstrTypeId*, *bstrProperty*);

Parameter	Beschreibung
<i>bstrTypeId</i>	Eine Zeichenfolge, die die ID des Typen oder des Planungstypen enthält.
<i>bstrProperty</i>	Eine Zeichenfolge, die den Namen der Eigenschaft enthält.

#### Rückgabewert

Der gegenwärtige Wert der geforderten Eigenschaft.

#### Beispiel

#### Beispiel

```
sub main(notused)
    Dim attributes(5)
    attributes(0) = "id"
    attributes(1) = "typename"
    attributes(2) = "real typename"
    attributes(3) = "caption singular"
    attributes(4) = "baseclass typename"
    Set Excel = CreateObject("excel.application")
    Excel.Workbooks.Add
    With Excel
        Rem Headline
        i = 1
        For k = 0 To 4
            .Cells(i, k + 1).Value = attributes(k)
        Next

        types = Config.GetAllPlanTypes(project_id, "", basetype)
        m = 0 ' TypeCounter
        i = 2 ' row counter

        dimx = UBound(types, 1)
        MsgBox (dimx+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 With
    sDatei = filename
    Excel.ActiveWorkbook.SaveAs sDatei
    Excel.Quit
    Set Excel = Nothing
    MsgBox "Done"

    Else
        MsgBox("No file selected.")
    end if
end sub
```



**Hinweis:** Der Typ-ID-Parameter dieser Funktion muß den eindeutigen Teil der POET-Objekt-ID der zu prüfenden Typen oder Planungstypen enthalten. Dieser ID beginnt mit der Zeichenfolge "cf", z. B. "cf2436". Diese POET-Objekt-Ids werden normalerweise von einem vorherigen Aufruf GetAllPlanTypes oder GetAllTypes zurückgegeben.

**3.8.1.6. GetAttributeInfoTN****SYNTAX**

GetAttributeInfoTN (BstrTypeName, bstrAttributeName, bstrProperty);

Parameter	Beschreibung
bstrTypeName	Eine Zeichenfolge, die den Typnamen eines Typs oder Planungstyps enthält.
bstrAttributeName	Eine Zeichenfolge, die den Namen des Attributes enthält.
bstrProperty	Eine Zeichenfolge, die den Namen der Eigenschaft enthält.

**Rückgabewert**

Der derzeitige Wert der Eigenschaft.

**Beispiel****Beispiel**

```
sub main(id)
    defval = Config.GetAttributeInfoTN("script", "name", "default-value")
    MsgBox(defval)
end sub
```

**3.8.1.7. GetAttributeInfo****SYNTAX**

GetAttributeInfo (bstrAttributeID, bstrProperty);

Parameter	Beschreibung
bstrAttributeID	Eine Zeichenfolge, die die ID des Attributes enthält.
bstrProperty	Eine Zeichenfolge, die den Namen der Eigenschaft enthält.

**Rückgabewert**

Der gegenwärtige Wert der geforderten Eigenschaft.

**Beispiel****Beispiel**

```
sub main(id)
    ...
    value = Config.value = Config.GetAttributeInfo(attr(m), "attribute-name")
    ...
end sub
```



**Hinweis:** Die Attributs-IDs können mit Hilfe eines vorherigen GetAllAttributes Aufrufs erhalten werden.

### 3.8.1.8. **GetAttributeValueList**

#### **SYNTAX**

GetAttributeVal ueLi st (bstrTypeName, bstrAttri buteName);

Parameter	Beschreibung
bstrTypeName	Eine Zeichenfolge, die den Namen des Typen oder des Planungstypen (GUID) enthält.
bstrAttributeName	Eine Zeichenfolge, die den Namen des Attributes enthält.

#### **Rückgabewert**

Ein Feld, welches die Wertepaare des Attributs enthält.

#### **Beispiel**

#### **Beispiel**

```
sub mai n(i d)
attrlist = Config. GetAttri buteVal ueLi st("script", "lan-
guagetype")
dimx = ubound(attrlist, 1)
for m=0 to dimx
    if attrlist(m) <> "" then
        value = Config. GetAttri buteVal uel nfo(attrli st(m), "val ue")
        MsgBox(val ue)
    end if
next
end sub
```



**3.8.1.9. GetAttributeValueInfo****SYNTAX**

GetAttributeValueInfo (bstrAttributeValueId, bstrProperty);

Parameter	Beschreibung
bstrAttributeValueId	Eine Zeichenfolge, die die ID des Eintrags der Attributwertliste enthält.
bstrProperty	Eine Zeichenfolge, die den Namen der Eigenschaft enthält.

**Rückgabewert**

Der Wert der Eigenschaft.

**Beispiel****Beispiel**

```
sub main(id)
    attrlist = Config.GetAttributeValueList("script", _
        languagetype")
    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
```



**Hinweis:** Normalerweise folgt GetAttributeValueInfo einem vorherigen Anruf für GetAttributeValueList

**3.8.1.10. GetParentPCRelations****SYNTAX**

GetParentPCRelations(bstrParentTypeName, bstrConstraint);

Parameter	Beschreibung
bstrParentTypeName	Eine Zeichenfolge, die den Namen des Vaters (Typen) enthält.
bstrConstraint	Wird nicht verwendet.

**Rückgabewert**

Ein Feld, das die IDs aller PC Relationen des Vaters (Typen) enthält.

**Beispiel****Beispiel**

```
sub main(id)
    ...
    pcrels = Config.GetParentPCRelations(parent_type, constraint)
    ...
end sub
```



**Hinweis:** Der Parameter **constraint** wird z. Z. nicht genutzt.

**3.8.1.11. GetPCRelations****SYNTAX**

GetPCRelations (bstrParentTypeName, bstrChildTypeName, bstrConstraint);

Parameter	Beschreibung
bstrParentTypeName	Eine Zeichenfolge, die den Namen des Vaters (Typen) enthält.
bstrChildTypeName	Eine Zeichenfolge, die den Namen des Kindes enthält.
bstrConstraint	Wird in dieser Version noch nicht verwendet.

**Rückgabewert**

Der Rückgabewert ist ein Feld, das die eindeutigen IDs aller Eltern-Kind-Beziehungen der angegebenen Vater- und Kindtypen enthält.

**Beispiel****Beispiel**

```
sub main(id)
...
parent_type = "ergocomplantdefault"
child_type = "ergocompprocessdefault"
pcrels = Config.GetPCRelations(parent_type, child_type, constraint)
...
end sub
```



**Note:** Der Parameter **constraint** wird z. Z. nicht genutzt

**3.8.1.12. GetPCRelationInfo****SYNTAX**

GetPCRelationInfo (bstrPCRelationId, bstrProperty);

Parameter	Beschreibung
bstrPCRelationId	Eine Zeichenfolge, die die ID der Eltern-Kind-Beziehung enthält.
bstrChildTypeName	Eine Zeichenfolge die den Namen der Eigenschaft enthält.

**Rückgabewert**

Der gegenwärtige Wert der Eigenschaft.

**Beispiel****Beispiel**

```
sub main(id)
...
value = Config.GetPCRelationInfo(pcrels(m), "id")
...
end sub
```



**Hinweis:** Diese Funktion wird normalerweise nach einem [GetParentPCRelations](#) oder [GetPCRelations](#) Aufruf ausgeführt.

**3.8.1.13. ResetItem****SYNTAX**

ResetItem();

**Rückgabewert**

keiner

**Beispiel**

```
sub main
...
Config.ResetItem
...
end sub
```



---

**Hinweis:** Diese Methode bringt das Skript in seinen Anfangszustand zurück. Alle statische Variablen werden gelöscht und mit einem Standardwert initialisiert. Es ist für alle XScriptItem Klassen implementiert. Die Methode wird jedes Mal implizit aufgerufen, wenn Sie ein Skript ausführen.

---

## 3.9. Klasse ScriptItemConvert

Methode	Beschreibung
<a href="#">Rtf2PlainText</a>	
<a href="#">TranslateText</a>	
ResetItem	Bringt das Skript in seinen Anfangszustand zurück

### 3.9.1. Liste aller XscriptItemConvert Methoden

#### 3.9.1.1. [Rtf2PlainText](#)

##### SYNTAX

Rtf2PlainText([bstrRtf](#));

Parameter	Beschreibung
<a href="#">bstrRtf</a>	Ein RTF formatierter Textstring.

##### Rückgabewert

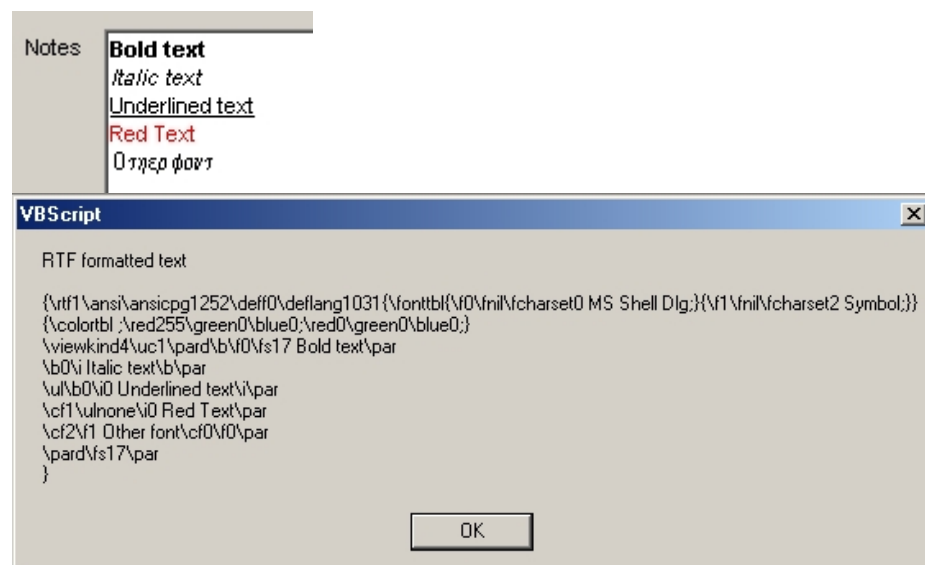
<a href="#">bstrPlainText</a>	Nur der Text ohne RTF Kopf, Formatierung und Steuerelemente.
-------------------------------	--

##### Beispiel

#### Beispiel

```
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
```



### 3.9.1.2. TranslateText

#### SYNTAX

TranslateText(*bstrText*);

Parameter	Beschreibung
<i>bstrText</i>	Der zu übersetzende Text.

#### Rückgabewert

<i>bstrTextTranslated</i>	Der übersetzte Text.
---------------------------	----------------------

#### Beispiel

#### Beispiel

```
sub main(notused)
orgText = "This is the Process Engineer VBScript documenta-
tion."
translatedText = Convert.TranslateText(orgText)
MsgBox("Original Text: " & orgText & vbCRLF & "Translated Text:
" & translatedText)
end sub
```

0407

german

This is the Process Engineer VBScript documentation.  
Das ist die Process Engineer VBScript Dokumentation.

*RLG File (for instance customized0407.rlg - German)*



## 3.10. Klasse ScriptItemVersion

Methode	Beschreibung
<b>Create</b>	Erzeugt eine neue Version.
<b>CreateDeep</b>	Erzeugt eine neue Version auch bei den Kindern.
<b>CheckOut</b>	Erzeugt eine Version, die sogleich weiterverwendet wird.
<b>CheckOutDeep</b>	Erzeugt bei dem aufgerufenen Objekt und seinen Kindern eine Version, die sogleich weiterverwendet wird.
<b>CanPlanningStateBeModified</b>	Whether the modification of component's planning state is allowed.
<b>SetPlanningState</b>	Setzt den Planungsstatus einer Version.
<b>GetPlanningState</b>	Gibt den Planungsstatus einer Version zurück.
<b>GetPlanningStates</b>	Gibt mögliche Planungsstature (globale und locale) zurück.
<b>GetFirstVersion</b>	Gibt die erste Version einer Komponente zurück.
<b>CreateReleaseTable</b>	Erzeugt eine Versionen Tabelle (Release Table). Zu Versionen Tabelle
<b>GetReleaseTable</b>	Zugriff auf eine vorhandenen Release Table.
<b>CreateSFI</b>	Erzeugt eine neue Version
<b>CreateDeepSFI</b>	Erzeugt neue Versionen rekursiv auch bei den Kindern
<b>CheckOutSFI</b>	Auschecken dieser Version
<b>CheckOutDeepSFI</b>	Rekursives Auschecken dieser Versionen
<b>Promote</b>	Promotes a component to a higher planning state.
<b>PDXMLFileCreated</b>	Wenn eine erweiterte Gültigkeit Teil dieser Version der Release Table ist, dann tritt die Methode PDXMLFileCreated bei der PDXML Datei ein.
<b>Use</b>	Verwendet eine existierende Version.
<b>UpdateRelations</b>	Update in Übereinstimmung mit den gegenwärtigen Filtereinstellung.
<b>ResetItem</b>	Setzt das Skript in den Anfangszustand zurück.

### 3.10.1. Liste aller XScriptItemVersion Methoden

#### 3.10.1.1. Create

##### SYNTAX

Create([bstrOldBDObjectID](#));

Parameter	Beschreibung
<a href="#">bstrOldBDObjectID</a>	Das Objekt, von dem eine neue Version erstellt werden soll.

##### Rückgabewert

Die Objekt ID der neu erzeugten Version.

##### Beispiel

#### Beispiel

```
REM Dieses Skript erzeugt und verwendet eine neue Version des
REM Objektes, auf dem es gestartet wird
Sub main(ol_d_sci_id)
    ol_d_base_id = Data.GetAttributebyId(ol_d_sci_id, "relationobject2")
    ol_d_base_name = Data.GetAttributebyId(ol_d_sci_id, "name")
    new_version_id = Version.Create(ol_d_base_id)

    If new_version_id <> "" then
        MsgBox("New version of <" & ol_d_base_name & "> created.")
        parent_id = Data.GetAttributebyId(ol_d_sci_id, "relationobject1")
        If parent_id <> "" then
            Call Version.Use(new_version_id, parent_id, ol_d_sci_id)
            MsgBox("New version of <" & ol_d_base_name & "> used.")
        End if
    Else
        MsgBox ("Checkout failed.")
    End if
End sub
```

### 3.10.1.2. CreateDeep

#### SYNTAX

CreateDeep(BSTR bstrOldBDOObjectID);

Parameters	Description
bstrOldBDOObjectID	Das Objekt, von dem eine neue Version erzeugt werden soll

#### Rückgabewert

Die Objekt ID der neuen Version.

#### Beispiel

#### Beispiel

```
REM Dieses Skript erzeugt und verwendet eine neue Version des
REM Objektes, auf dem es gestartet wird

sub main(ol d_sci _i d)
    ol d_base_id = Data.GetAttributebyId(ol d_sci _i d, "relationobject2")
    ol d_base_name = Data.GetAttributebyId(ol d_sci _i d, "name")
    new_version_id = Version.CreateDeep(ol d_base_id)
    If new_version_id <> "" then
        MsgBox("New version of <" & ol d_base_name & "> created.")
        parent_id = Data.GetAttributebyId(ol d_sci _i d, "relationobject1")
        If parent_id <> "" then
            call Version.Use(new_version_id, parent_id, ol d_sci _i d)
            MsgBox("New version of <" & ol d_base_name & "> used.")
        End if
    Else
        MsgBox ("Checkout failed.")
    End if
end sub
```

Die Funktionsweise von *CreateDeep* ist die Gleiche wie bei *Create* beschrieben. Zum Unterschied zu *Create* erhalten die **Kinder** der ausgewählten Komponente ebenfalls eine neue Version.



**3.10.1.3. CheckOut****SYNTAX**

**CheckOut** (*bstrOldBDObjectID*, *bstrParentObjectID*,  
*bstrOldUsageObjectID*);

Parameter	Beschreibung
<i>bstrOldBDObjectID</i>	Das Objekt, von dem eine neue Version erstellt werden soll.
<i>bstrParentObjectID</i>	Die Objekt ID des Vaters, in dessen Kinderliste die alte Version durch eine neue Version ersetzt werden soll.
<i>bstrOldUsageObjectID</i>	Die Objekt ID des gebrauchten Objekts der alten Version (sub-compitem, relationship object).

**Rückgabewert**

Die Objekt-ID der neuen Version.

**Beispiel****Beispiel**

```

REM Diese Methode überprüft die neue Version eines Objektes,
REM auf dem das Skript gestartet wurde.
REM Überprüfen heißt hier: eine neue Version erstellen und die-
REM se benutzen.
REM Hinweis: Die Eingangsparameter der Version.CheckOut methode
REM müssen den Regeln folgen:
REM old_base_id : Ergocompbase des Objekts, auf dem das Skript
REM gestartet wurde
REM old_sci_id : Das Objekt (subcompitem, relationship-Objekt),
REM auf dem das Skript gestartet wurde
REM parent_id : Der Vater des Objekts, auf dem das Skript ge-
REM startet wurde
REM Dasselbe gilt für die Relationship-Objekte

Sub main(old_sci_id)
    parent_id = Data.GetAttributebyId(old_sci_id, "relationobject1")
    old_base_id = Data.GetAttributebyId(old_sci_id, "ergocompbase")
    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

```

**3.10.1.4. CheckOutDeep****SYNTAX**

CheckOutDeep([bstrOldBDOObjectId](#), [bstrParentObjectId](#),  
[bstrOldUsageObjectId](#));

Parameters	Description
<a href="#">bstrOldBDOObjectId</a>	Das Objekt, von dem eine neue Version erstellt werden soll.
<a href="#">bstrParentObjectId</a>	Die Objekt ID des Vaters, in dessen Kinderliste die alte Version durch eine neue Version ersetzt werden soll.
<a href="#">bstrOldUsageObjectId</a>	Die Objekt ID des verwendeten Objekts der alten Version (subcompitem, relationship object).

**Rückgabewert**

Die Objekt ID der neuen Version.

**Beispiel****Beispiel**

```
REM Diese Methode überprüft die neue Version eines Objektes,
REM auf dem das Skript gestartet wurde.
REM Überprüfen heißt hier: eine neue Version erstellen und die-
REM se benutzen.
REM Hinweis: Die in-parameter der Version.CheckOut methode müs-
REM sen den Regeln folgen:
REM old_base_id : Ergocompbase des Objekts, auf dem das Skript
REM gestartet wurde
REM old_sci_id : Das Objekt (subcompitem, relationship-Objekt),
REM auf dem das Skript gestartet wurde
REM parent_id : Der Vater des Objekts, auf dem das Skript ge-
REM startet wurde
REM Dasselbe gilt für die Relationship-Objekte

Sub main(old_sci_id)
    parent_id = Data.GetAttributebyId(old_sci_id, "relationship1")
    old_base_id = Data.GetAttributebyId(old_sci_id, "relationship2")
    old_base_name = Data.GetAttributebyId(old_base_id, "name")
    new_version_id = Version.CheckOutDeep(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
```



**Hinweis:** Diese Methode arbeitet ähnlich wie [CheckOut](#), jedoch werden die Kinder (Rekursion) auch automatisch versioniert. CheckOut verwendet ein [CreateDeep](#) und eine anschließende Verwendung ([Use](#)).

**3.10.1.5. CanPlanningStateBeModified****SYNTAX**

CheckOut (bstrOldBDOObjectId);

Parameters	Description
bstrOldBDOObjectId	The component whose modifiability is to check.

**Rückgabewert**

Die Objekt ID der neuen Version.

**Beispiel****Beispiel**

```
REM This script checks a components' modifiability
Sub main(ol_d_sci_id)
    ol_d_base_id = Data.GetAttributeById(ol_d_sci_id, "ergocompbase")
    ol_d_base_name = Data.GetAttributeById(ol_d_sci_id, "name")
    modifiable = Version.CanPlanningStateBeModified(ol_d_base_id)
    MsgBox(ol_d_base_name & ": Modifiability is " & modifiable)
End sub
```

### 3.10.1.6. SetPlanningState

#### SYNTAX

SetPlanningState(bstrObjectId, bstrPlanningStateObjectId);

Parameter	Beschreibung
bstrObjectId	Die Version, deren Planungsstatus gesetzt oder zurückgesetzt werden soll.
bstrPlanningStateObjectId	Die Objekt ID des Planungsstatus eines Objekts, zu dem die Version gesetzt werden soll.

#### Rückgabewert

kein Rückgabewert

#### Beispiel

#### Beispiel

```
REM Setzen des Planungsstatus einer Komponente
REM Rufen Sie dieses Skript auf eine Komponente auf, deren Planungsstatus Sie (neu) setzen wollen.
REM Vergewissern Sie sich, dass sie den richtigen Planungsstatus benutzen, indem sie ein zwingendes/eindeutiges Attribut benutzen.
Sub main (id)
    object_id=Data.GetAttributeById(id, "relationobject2")
    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", "dodefaultimpl", "=", 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
```

**3.10.1.7. GetPlanningState****SYNTAX**

GetPlanningState(*bstrObjectId*);

Parameter	Beschreibung
<i>bstrObjectId</i>	Die Version, deren Planungsstatus erhalten werden soll.

**Rückgabewert**

Die Objekt ID des überprüften Planungsstatus.

**Beispiel****Beispiel**

```
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
```

**3.10.1.8. GetPlanningStates****SYNTAX**

GetPlanningStates(*bstrObjectId*, *bstrMode*, *bstrScope*);

Parameter	Beschreibung
<i>bstrObjectId</i>	Jede Objekt ID innerhalb des Projekts.
<i>bstrMode</i>	Beschränkt den zurückgegebenen Planungsstatus auf einen Bereich oder eine höhere Ebene.
<i>bstrScope</i>	Empfängt die Planungsstature global oder local.

Mögliche Werte von ***bstrMode***

"ALL"	Erhält alle Planungsstature.
"WORKING"	Erhält nur Planungsstature vom Status "working".
"INTEGRATE"	Erhält nur Planungsstature vom Status "integrate".
"RELEASED"	Erhält nur Planungsstature vom Status "released".
"HIGHER_LEVEL"	Erhält nur Planungsstature vom Status "höher" als der gegenwärtige.

Mögliche Werte von ***bstrScope***

"LOCAL"	Erhält nur den Planungsstatus von der Projektbibliothek.
"GLOBAL"	Erhält nur den Planungsstatus von der Bibliothek.

**Rückgabewert**

Ein Feld, das die ID's der enthaltenen Objekte beinhaltet.

**Beispiel****Beispiel**

```

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

```

**3.10.1.9. GetFirstVersion****SYNTAX**

GetFirstVersion(*bstrObjectId*);

Parameter	Beschreibung
<i>bstrObjectId</i>	Die Objekt ID des Objektes, von dem die erste Version erhalten werden soll.

**Rückgabewert**

Die Objekt ID der ersten Version.

**Beispiel****Beispiel**

```
REM Erhält die erste Version eines Objekts
REM Dieses Skript muss auf einem subcompitem oder Relationship-
Objekt aufgerufen werden.
Sub main(ol_d_sci_id)
  ol_d_base_id = Data.GetAttributebyId(ol_d_sci_id, "relationobject2")
  If ol_d_base_id <> "" then
    ol_d_parent_id = Data.GetAttributebyId(ol_d_sci_id, "relationobject1")
    If ol_d_parent_id <> "" then
      first_version_id=Version.GetFirstVersion(ol_d_base_id)
      If first_version_id <> "" then
        call Version.Use(first_version_id, ol_d_parent_id, ol_d_sci_id)
        MsgBox("First Version used.")
      End if
    End if
  End if
End sub
```

### 3.10.1.10. CreateReleaseTable

#### SYNTAX

CreateReleaseTable (bstrOldBDOObjectId);

Parameter	Beschreibung
bstrOldBDOObjectId	Das Objekt, von dem eine neue Tabelle erzeugt werden soll.

#### Rückgabewert

Die Objekt ID der neu erzeugten Versionen Tabelle (ReleaseTable).

#### Beispiel

#### Beispiel

```
REM This script creates and uses a new release table of the  
Rem object from which the script is executed
```

```
Sub main(ol_d_sci_id)
```

```
  ol_d_base_id = Data.GetAttributebyId(ol_d_sci_id, "relationobject2")
```

```
  ol_d_base_name = Data.GetAttributebyId(ol_d_sci_id, "name")
```

```
  new_release_table_id = Versi on. CreateReleaseTable(ol_d_base_id)
```

```
  If new_release_table_id <> "" then
```

```
    MsgBox("New release table of <" & ol_d_base_name & "> created.")
```

```
  Else
```

```
    MsgBox("Creation failed.")
```

```
  End if
```

```
End sub
```



#### Hinweis:

Wenn es bereits eine release table für eine Version des Objektes gibt, dann wird stattdessen die vorhandene Tabelle zurückgegeben und ein Hinweis in die Log-Datei eingetragen.

---



### 3.10.1.11. GetReleaseTable

**SYNTAX**

GetReleaseTable (bstrOldBDOObjectID);

Parameter	Beschreibung
bstrOldBDOObjectID	Das Objekt von dem die release table erhalten werden soll.

**Rückgabewert**

Die Objekt ID der release table.

**Beispiel****Beispiel**

```
REM This script gets the existing release table of the object
REM from which the script is executed
Sub main(ol_d_sci_id)
    old_base_id = Data.GetAttributebyId(ol_d_sci_id, "relationobject2")
    old_base_name = Data.GetAttributebyId(ol_d_sci_id, "name")
    existing_releasetable_id = Version.GetReleaseTable(ol_d_base_id)
    if existing_releasetable_id <> "" then
        MsgBox("Existing Release Table is " & existing_releasetable_id & " .")
    else
        MsgBox("GetReleaseTable failed.")
    end if
End sub
```

### 3.10.1.12. CreateSFI

**SYNTAX**

CreateSFI (bstrOldBDOObjectID);

Parameter	Beschreibung
bstrOldBDOObjectID	Das Objekt, von dem eine neue Version erzeugt werden soll.

**Rückgabewert**

Die Objekt ID einer neu erzeugten Version.

**Beispiel****Beispiel**

```
REM This script creates and uses a new version of the object
REM from which the script is executed
Sub main(ol_d_sci_id)
    old_base_id = Data.GetAttributebyId(ol_d_sci_id, "relationobject2")
    old_base_name = Data.GetAttributebyId(ol_d_sci_id, "name")
    new_version_id = Version.CreateSFI(ol_d_base_id)

    if new_version_id <> "" then
        MsgBox("New version of <" & old_base_name & "> created.")
        parent_id = Data.GetAttributebyId(ol_d_sci_id, "relationobject1")
        if parent_id <> "" then
            call Version.Use(new_version_id, parent_id, ol_d_sci_id)
            MsgBox("New version of <" & old_base_name & "> used.")
        end if
    else
        MsgBox("Checkout failed.")
    end if
End sub
```

**3.10.1.13. CreateDeepSFI****SYNTAX****CreateSFI** (*bstrOldBDOObjectId*);

Parameter	Beschreibung
<i>bstrOldBDOObjectId</i>	Das Objekt, von dem eine neue Version erzeugt werden soll.

**Rückgabewert**

Die Objekt ID einer neu erzeugten Version.

**Beispiel****Beispiel**

```
REM This script creates and uses a new version of the object
REM from which the script is executed
Sub main(ol_d_sci_id)
  ol_d_base_id = Data.GetAttributeById(ol_d_sci_id, "relationobject2")
  ol_d_base_name = Data.GetAttributeById(ol_d_sci_id, "name")
  new_version_id = Version.CreateDeepSFI(ol_d_base_id)

  if new_version_id <> "" then
    MsgBox("New version of <" & ol_d_base_name & "> created.")
    parent_id = Data.GetAttributeById(ol_d_sci_id, "relationobject1")
    if parent_id <> "" then
      call Version.Use(new_version_id, parent_id, ol_d_sci_id)
      MsgBox("New version of <" & ol_d_base_name & "> used.")
    end if
  else
    MsgBox("Checkout failed.")
  end if
End sub
```

**3.10.1.14. CheckOutSFI****SYNTAX**

CheckOutSFI (bstrOldObjectId,  
bstrParentObjectId,  
bstrOldUsageObjectId);

Parameter	Beschreibung
bstrOldBDOObjectId	Das Objekt, von dem eine neue Version erzeugt werden soll.
bstrParentObjectId	Die Objekt ID von den Eltern in dessen Kinderliste die alte Version durch eine neue ersetzt werden sollte.
bstrOldUsageObjectId	Die Object ID, von der alten Version eines usage object (subcompitem, relationship object).

**Rückgabewert**

Die Objekt ID von einer neu erzeugten Version.

**Beispiel****Beispiel**

```

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 ver-
REM sion.
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(ol_d_sci_id)
    parent_id = Data.GetAttributebyId(ol_d_sci_id, "relationship1")
    ol_d_base_id = Data.GetAttributebyId(ol_d_sci_id, "relationship2")
    ol_d_base_name = Data.GetAttributebyId(ol_d_base_id, "name")

    new_version_id = Version.CheckOutSFI(ol_d_base_id, parent_id, ol_d_sci_id)
    if new_version_id <> "" then
        MsgBox("Checked out a new version of <" & ol_d_base_name & ">.")
    else
        MsgBox("Checkout failed.")
    end if
End sub

```

**3.10.1.15. CheckOutDeepSFI****SYNTAX**

CheckOutDeepSFI (bstrOldObjectId,  
bstrParentObjectId,  
bstrOldUsageObjectI);

Parameter	Beschreibung
bstrOldBDOObjectId	Das Objekt, von dem eine neue Version erzeugt werden soll.
bstrParentObjectId	Die Objekt ID des Vaters, in dessen Kinderliste die alte Version durch eine neue Version ersetzt werden soll.
bstrOldUsageObjectId	Die Objekt ID des verwendeten Objekts der alten Version (subcompitem, relationship object).

**Rückgabewert**

Die Objekt ID von einer neu erzeugten Version.

**Beispiel****Beispiel**

```

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 ver-
REM sion.
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, "relationobject2")
    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

```

**3.10.1.16. Promote**

This method promotes a component to a higher planning state.

**SYNTAX**

Promote();

Parameter	Beschreibung
keine	-

**Rückgabewert**

keiner

**Beispiel****Beispiel**

```
Sub main(i d)
    ..
    Version. Promote(i d)
    ..
End sub
```

**3.10.1.17. PDXMLFileCreated****SYNTAX**

PDXMLFileCreated (bstrOldBDOObjectID,  
bstrExtendedEffectivity,  
bWasCreated);

Parameter	Beschreibung
bstrOldBDOObjectID	Das Objekt, von dem eine neue Version erzeugt werden soll.
bstrExtendedEffectivity	Die erweiterte Gültigkeit.
bWasCreated	"True"

**Rückgabewert**

keiner

**Beispiel****Beispiel**

```
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, "relationobject2")
    if old_base_id <> "" then
        old_parent_id = Data.GetAttributeById(old_sci_id, "relationobject1")
        if old_parent_id <> "" then
            call Version.PDXMLFileCreated(old_base_id, "{2-10}", true)
        end if
    end if
End sub
```

**3.10.1.18. Use****SYNTAX**

```
Use(bstrNewBDObjectld, bstrParentObjectld,
bstrOldUsageObjectld);
```

Parameter	Beschreibung
bstrNewBDObjectld	Die neue Version eines Objekts, die anstatt der alten Version benutzt werden soll.
bstrParentObjectld	Die Objekt ID eines Vaters, in dessen Kinderliste die alte Version durch eine neue ersetzt werden soll.
bstrOldUsageObjectld	Die Objekt ID des gebrauchten Objekts (subcompitem, relationship object) der alten Version.

**Rückgabewert**

kein Rückgabewert

**Beispiel****Beispiel**

```
REM Dieses Skript erzeugt und verwendet eine neue Version des
REM Objektes, auf dem es gestartet wird
```

```
Sub main(ol_d_sci_id)
```

```
  ol_d_base_id = Data.GetAttributebyld(ol_d_sci_id, "relationobject2")
```

```
  ol_d_base_name = Data.GetAttributebyld(ol_d_sci_id, "name")
```

```
  new_version_id =Version.Create(ol_d_base_id)
```

```
  If new_version_id <> "" then
```

```
    MsgBox("New version of <" & ol_d_base_name & "> created.")
```

```
    parent_id = Data.GetAttributebyld(ol_d_sci_id, "relationobject1")
```

```
    If parent_id <> "" then
```

```
      Call Version.Use(new_version_id, parent_id, ol_d_sci_id)
```

```
      MsgBox("New version of <" & ol_d_base_name & "> used.")
```

```
    End if
```

```
  Else
```

```
    MsgBox ("Checkout failed.")
```

```
  End if
```

```
End sub
```

### 3.10.1.19. UpdateRelations

#### SYNTAX

UpdateRelations(*bstrObjectId*);

Parameter	Beschreibung
<i>bstrObjectId</i>	Das Objekt, dessen Strukturen aktualisiert werden sollen nach der Filtereinstellung.
<i>bstrExtendedEffectivity</i>	Die gegebene erweiterte Gültigkeit.

#### Rückgabewert

kein Rückgabewert

#### Beispiel

#### Beispiel

```
REM Get the first version of an object
REM This script must be called on a subcomponent 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(ol_d_sci_id)
```

```
    old_base_id = Data.GetAttributebyId(ol_d_sci_id, "ergocompbase")
    if old_base_id <> "" then
        old_parent_id = Data.GetAttributebyId(ol_d_sci_id, "relationship1")
        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, ol_d_sci_id)
                MsgBox("First Version used.")
                call Version.UpdateRelations(first_version_id, "#DROP(R(2))")
            end if
        end if
    end if
end sub
```

## 3.11. Klasse ScriptItemUnit

Methode	Beschreibung
<b>Convert</b>	Konvertiert einen Wert von der Quell- zur Zieleinheit.
<b>GetDefaultUnitByAttr</b>	Gibt die Standard-Einheit des Typnamens und des Attributnamens (Schlüssel) zurück.
<b>GetDefaultUnitByCat</b>	Gibt die Standard-Einheit der Einheitenkategorie zurück.
<b>GetDefaultUnitByUnit</b>	Gibt die Standard-Einheit anhand einer anderen Einheit dieser Kategorie zurück.
<b>GetUIDefaultUnitByAttr</b>	Gibt die angezeigte Standard Einheit anhand des Typnamens und des Attributnamens (Schlüssel) zurück.
<b>GetUIDefaultUnitByCat</b>	Gibt die angezeigte Standard Einheit anhand der Kategorie zurück.
<b>GetUIDefaultUnitByUnit</b>	Gibt die angezeigte Standard Einheit anhand einer anderen Einheit dieser Kategorie zurück
<b>GetAllUnitsByAttr</b>	Gibt ein Feld von allen Einheiten einer Kategorie anhand des Namens des Typen und des Attributnamen (Schlüssel) zurück.
<b>GetAllUnitsByCat</b>	Gibt ein Feld von allen Einheiten einer Kategorie zurück.
<b>ConvertToDefaultUnit</b>	Konvertiert einen Wert von der Quell- zur Standardeinheit.
<b>ConvertToUIDefaultUnit</b>	Wandelt den Quellwert in die sichtbare Standardeinheit um.
<b>ResetItem</b>	Setzt das Skript in den Anfangszustand zurück.

### 3.11.1. Liste aller XScriptItemUnit Methoden

#### 3.11.1.1. Convert

##### SYNTAX

`Convert(bstrSourceUnit, bstrTargetUnit, vSourceValue);`

Parameter	Beschreibung
<code>bstrSourceUnit</code>	Die Ausgangs-Einheit.
<code>bstrTargetUnit</code>	Die Ziel-Einheit.
<code>vSourceValue</code>	Der Wert vor der Umwandlung in die Zieleinheit.

##### Script Return Value

<code>vTargetValue</code>	Der Wert nach der Konvertierung.
---------------------------	----------------------------------

##### Beispiel

#### Beispiel

```
sub main(i d)
...
currentunit = "UA_TIME_DAY"
selectedunit = "UA_TIME_SEC"
inputvalue = 1
outputvalue = Unit.Convert (currentunit, selectedunit, inputvalue)
...
end sub
```



**3.11.1.2. GetDefaultUnitByAttr****SYNTAX**

GetDefaultUnitByAttr(*bstrTypeName*, *bstrAttributeName*);

Parameter	Beschreibung
<i>bstrTypeName</i>	Der eindeutige Typenname.
<i>bstrAttributeName</i>	Der Name eines Attributes dieses Typs (der Typenname UND der Name des Attributes definieren die Eindeutigkeit).

**Rückgabewert**

<i>bstrDefUnit</i>	Die Standard-Einheit (z. B. "UA_TIME_SEC").
--------------------	---

**Beispiel****Beispiel**

```
Sub main(i d)
```

```
...
```

```
type_name = "ergocompprocessdefault"
```

```
attribute_name = "time"
```

```
def_unit = Unit.GetDefaultUnitByAttr(type_name, attribute_name)
```

```
...
```

```
end sub
```



**Hinweis:** Jedes Attribut gehört zu einer Kategorie (z. B. "UA\_TIME"). Die Standard-Einheit (z. B. "UA\_TIME\_SEC") wird in der Konfiguration festgelegt. Angezeigt werden aber die Einheiten, die individuell im Eigenschaftsdialog oder im Browser festgelegt wurden (z. B. "UA\_TIME\_MIN").

**3.11.1.3. GetDefaultUnitByCat****SYNTAX**

GetDefaultUnitByCat(*bstrCategory*);

Parameter	Beschreibung
<i>bstrCategory</i>	Die Einheiten Kategorie (z. B. "UA_TIME").

**Rückgabewert**

<i>bstrDefUnit</i>	Die Standard-Einheit (z. B. "UA_TIME_SEC").
--------------------	---

**Beispiel****Beispiel**

```
sub main(i d)
```

```
...
```

```
unitcategory = "UA_TIME"
```

```
def_unit = Unit.GetDefaultUnitByCat(unitcategory)
```

```
...
```

```
end sub
```



**Hinweis:** Jedes Attribut gehört zu einer Kategorie (z. B. "UA\_TIME"). Die Standard-Einheit (z. B. "UA\_TIME\_SEC") wird in der Konfiguration festgelegt. Angezeigt werden aber die Einheiten, die individuell im Eigenschaftsdialog oder im Browser festgelegt wurden (z. B. "UA\_TIME\_MIN").

**3.11.1.4. GetDefaultUnitByUnit****SYNTAX**

GetDefaultUnitByUnit(*bstrUnit*);

Parameter	Beschreibung
<i>bstrUnit</i>	Jede andere Einheit (z. B. "UA_TIME_MIN"), die zu derselben Einheitenkategorie (z. B. "UA_TIME") gehört.

**Rückgabewert**

<i>bstrDefUnit</i>	Die Standard-Einheit (z. B. UA_TIME_SEC).
--------------------	---

**Beispiel****Beispiel**

```
Sub main(i d)
...
currentunit = "UA_TIME_DAY"
def_unit = Unit.GetDefaultUnitByUnit(currentunit)
...
End sub
```



**Hinweis:** Jedes Attribut gehört zu einer Kategorie (z. B. "UA\_TIME"). Die Standard-Einheit (z. B.. "UA\_TIME\_SEC") wird in der Konfiguration festgelegt. Angezeigt werden aber die Einheiten, die individuell im Eigenschaftsdialog oder im Browser festgelegt wurden (z. B.. "UA\_TIME\_MIN").

**3.11.1.5. GetUIDefaultUnitByAttr****SYNTAX**

GetUIDefaultUnitByAttr(*bstrTypeName*, *bstrAttributeName*);

Parameter	Beschreibung
<i>bstrTypeName</i>	Der eindeutige Typenname.
<i>bstrAttributeName</i>	Der Name eines Attributes dieses Typs.

**Rückgabewert**

<i>bstrDefUnit</i>	Die angezeigte Einheit (z. B. "UA_TIME_SEC").
--------------------	---

**Beispiel****Beispiel**

```
Sub main(i d)
...
type_name = "ergocompprocessdefault"
attribute_name = "time"
def_unit = Unit.GetUIDefaultUnitByAttr(type_name, attribute_name)
...
end sub
```



**Hinweis:** Jedes Attribut gehört zu einer Kategorie (z. B. "UA\_TIME"). Die Standard-Einheit (z. B.. "UA\_TIME\_SEC") wird in der Konfiguration festgelegt. Ange-

zeigt werden aber die Einheiten, die individuell im Eigenschaftsdialog (UI) oder im Browser festgelegt wurden (z. B.. "UA\_TIME\_MIN").

### 3.11.1.6. GetUIDefaultUnitByCat

#### SYNTAX

GetUI Defaul tUni tByCat(*bstrCategory*);

Parameter	Beschreibung
<i>bstrCategory</i>	Die Einheiten-Kategorie (z. B. "UA_TIME").

#### Rückgabewert

<i>bstrDefUnit</i>	Die angezeigte Einheit (z. B. "UA_TIME_SEC").
--------------------	---

#### Beispiel

#### Beispiel

```
Sub mai n(i d)
...
uni tcategory = "UA_TIME"
def_uni t = Uni t. GetUI Defaul tUni tbyCat(uni tcategory)
...
end sub
```



**Hinweis:** Jedes Attribut gehört zu einer Kategorie (z. B. "UA\_TIME"). Die Standard-Einheit (z. B.. "UA\_TIME\_SEC") wird in der Konfiguration festgelegt. Angezeigt werden aber die Einheiten, die individuell im Eigenschaftsdialog (UI) oder im Browser festgelegt wurden (z. B.. "UA\_TIME\_MIN").

### 3.11.1.7. GetUIDefaultUnitByUnit

#### SYNTAX

GetUI Defaul tUni tByUni t(*bstrUnit*);

Parameter	Beschreibung
<i>bstrUnit</i>	Jede andere Einheit (z. B. "UA_TIME_MIN"), die zu derselben Einheitenkategorie (z. B. "UA_TIME") gehört.

#### Rückgabewert

<i>bstrDefUnit</i>	Die Standard-Einheit (z. B. UA_TIME_SEC").
--------------------	--

#### Beispiel

#### Beispiel

```
Sub mai n(i d)
...
currentuni t = "UA_TIME_DAY"
def_uni t = Uni t. GetUI Defaul tUni tbyUni t(currentuni t)
...
End sub
```



**Hinweis:** Jedes Attribut gehört zu einer Kategorie (z. B. "UA\_TIME"). Die Standard-Einheit (z. B.. "UA\_TIME\_SEC") wird in der Konfiguration festgelegt. Ange-

zeigt werden aber die Einheiten, die individuell im Eigenschaftsdialog (UI) oder im Browser festgelegt wurden (z. B.. "UA\_TIME\_MIN").

### 3.11.1.8. GetAllUnitsByAttr

#### SYNTAX

GetAllUnitsByAttr(*bstrTypeName*, *bstrAttribute*);

Parameter	Beschreibung
<i>bstrTypeName</i>	Der eindeutige Typenname.
<i>bstrAttribute-Name</i>	Der Name eines Attributes dieses Typs (die Typbezeichnung UND der Attributname ergeben die ID).

#### Rückgabewert

<i>pUnits</i>	An array of pairs of visible unit name (e.g. "seconds") and internal unit name (e.g. "UA_TIME_SEC").
---------------	--

#### Beispiel

#### Beispiel

```
sub main(id)
...
type_name = "ergocompprocessdefault"
attribute_name = "time"
units = Unit.GetAllUnitsByAttr(type_name, attribute_name)
Set d = CreateObject("Scripting.Dictionary")
dim units = ubound(units, 1)
REM write the visible/internal values to a VB dictionary
if dim units >= 0 then
for m = 0 to dim units
d.Add units(m, 0), units(m, 1)
next
end if
...
end sub
```



**Hinweis:** Bei anderen Funktionsaufrufen wird der interne Name der Einheit (z. B. "UA\_TIME\_SEC") verwendet und nicht der sichtbare Name der Einheit (z. B. "Sekunden").

**3.11.1.9. GetAllUnitsByCat****SYNTAX**

GetAllUnitsByCat(*bstrCategory*);

Parameter	Beschreibung
<i>bstrCategory</i>	Die Einheiten Kategorie (z. B. "UA_TIME").

**Script Return Value**

<i>pUnits</i>	Ein Feld mit dem sichtbaren Name der Einheit und (z. B. "Sekunden") und dem internen Name (z. B. "UA_TIME_SEC").
---------------	--

**Beispiel****Beispiel**

```
sub main(id)
...

unitcategory = "UA_TIME"

units = Unit.GetAllUnitsByCat(unitcategory)

Set d = CreateObject("Scripting.Dictionary")
dim units = ubound(units, 1)
REM write the visible/internal values to a VB dictionary
if dim units >= 0 then
for m = 0 to dim units
d.Add units(m, 0), units(m, 1)
next
end if
...
end sub
```



---

**Hinweis:** Bei anderen Funktionsaufrufen wird der interne Name der Einheit (z. B. "UA\_TIME\_SEC") verwendet und nicht der sichtbare Name der Einheit (z. B. "Sekunden").

---

**3.11.1.10. ConvertToDefaultUnit****SYNTAX**

ConvertToDefaultUnit(*bstrSourceUnit*, *vSourceValue*);

Parameter	Beschreibung
<i>bstrSourceUnit</i>	Die Anfangseinheit.
<i>vSourceValue</i>	Der Wert vor der Umwandlung in die Standardeinheit.

**Script Return Value**

<i>vTargetValue</i>	Der Wert nach der Konvertierung.
---------------------	----------------------------------

**Beispiel****Beispiel**

```
sub main(id)
...
currentunit = "UA_TIME_DAY"
inputvalue = 1
outputvalue = Unit.ConvertToDefaultUnit (currentunit, input-
value)
...
end sub
```

**3.11.1.11. ConvertToUIDefaultUnit****SYNTAX**

ConvertToUIDefaultUnit(*bstrSourceUnit*, *vSourceValue*);

Parameter	Beschreibung
<i>bstrSourceUnit</i>	Die Ausgangs-Einheit.
<i>vSourceValue</i>	Der Wert vor der Umwandlung in die Standardeinheit.

**Script Return Value**

<i>vTargetValue</i>	Der Wert nach der Konvertierung.
---------------------	----------------------------------

**Beispiel****Beispiel**

```
sub main(notused)
unitcategory = "UA_TIME"
currentunit = "UA_TIME_DAY"
currentunitname = "d"

if unitcategory <> "" then
units = Unit.GetAllUnitsByCat(unitcategory)

Dim combo
Set d = CreateObject("Scripting.Dictionary")
dimunits = ubound(units, 1)
REM write the visible/internal values to a VB dictionary
if dimunits >= 0 then
Redim combo(dimunits, 1)
for m = 0 to dimunits
d.Add units(m, 0), units(m, 1)
combo(m, 0) = units(m, 1)
combo(m, 1) = units(m, 0)
next
next
```

```

call Dialog.CreateInputControl("1", "ComboBox", "Unit _
Selection", combo)
call Dialog.CreateInputControl("2", "EditString", "Input _
[" & currentunitname & "]", "1")
retval = Dialog.InputBox("Select a Unit for conversion")
if retval = True then
    selectedunit = Dialog.GetInputControlValue("1")
    inputvalue = Dialog.GetInputControlValue("2")
end if

outputvalue = Unit.Convert ( currentunit, selectedunit, _
inputvalue)
text = inputvalue & " (" & d.Item(currentunit) & ") --> _
" & outputvalue & " (" & d.Item(selectedunit) & ")"
call Dialog.MessageBox("Conversion to Selected Unit", _
text)

def_unit = Unit.GetDefaultUnitByCat(unitcategory)
ui_def_unit = Unit.GetUI DefaultUnitByCat(unitcategory)

REM alternative
REM def_unit = Unit.GetDefaultUnitByUnit(currentunit)
REM ui_def_unit = Unit.GetUI DefaultUnitByUnit(currentunit)

outputvalue = Unit.ConvertToDefaultUnit(currentunit, _
inputvalue)
text = inputvalue & " (" & d.Item(currentunit) & ") --> _
" & outputvalue & " (" & d.Item(def_unit) & ")"
call Dialog.MessageBox("Conversion to Default Unit", _
text)

if def_unit <> ui_def_unit then
    outputvalue = Unit.ConvertToUI DefaultUnit(currentunit, _
inputvalue)
    text = inputvalue & " (" & d.Item(currentunit) & ") _
--> " & outputvalue & " (" & d.Item(ui_def_unit) & ")"
    call Dialog.MessageBox("Conversion to User Interface _
Default Unit", text)
end if

end if
else
    MsgBox("No available units")
end if

end sub

```

## 3.12. Klasse ScriptItemLock

Methode	Bedeutung
<b>LockObject</b>	Damit werden Objekte verwendet (Gelockt).
<b>UnlockObject</b>	Verwendung aufheben
<b>UnlockAllObject</b>	Gibt aller Objekte auf einmal frei, die zuvor mit LockObject gelockt wurden. Die IDs aller gelockten Objekten werden temporär in einer Liste gespeichert. Der Funktionsaufruf <i>UnlockAllObject</i> bereinigt auch diese Liste.
<b>IsObjectLocked</b>	Prüft ob ein Objekt verwendet wird oder nicht.
<b>GetObjectLockInfo</b>	Gibt Informationen zu der Verwendungsart (Lock _ Status).
<b>ResetItem</b>	Zurücksetzen des Skriptes in seinen Anfangsstatus.



**Hinweis:** Bei der Verwendung der Klasse *ScriptItemLock* soll nicht fälschlicherweise der Eindruck entstehen, dass es möglich wäre, Objekte, die von anderen Benutzern bearbeitet werden, zur Bearbeitung wieder freizugeben. Dies ist natürlich nicht möglich. Außerdem sind Verwendungen (Locks) nicht persistent. Sie können nur Objekte freigeben, die innerhalb der Transaktion (Client Prozess) bearbeitet wurden, in der das Skript aufgerufen wird.



### 3.12.1. Liste aller XScriptItemLock Methoden

#### 3.12.1.1. LockObject

##### Syntax

LockObject(*bstrObjectId*, *bstrLockMode*);

Parameter	Beschreibung
<i>bstrObjectId</i>	Das verwendete Objekt.
<i>bstrLockMode</i>	Der Modus, der für die Komponente verwendet werden soll (siehe Werte für <i>bstrLockMode</i> ).

Mögliche Werte für ***bstrLockMode*** sind

- NOT\_DELETE
- NOT\_LINK
- READ
- LINK
- WRITE
- DELETE

##### Rückgabewert

für *bFlag*

- True (-1): Falls der Befehl erfolgreich gewesen ist.
- False (0): Sonst (die Komponente wird z. B. schon von jemand anderem verwendet).

##### Beispiel

#### Beispiel

```
sub main(i d)
  base_id = Data.GetAttributebyId(i d, "relationobject2")
  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
```

### 3.12.1.2. UnlockObject

#### Syntax

UnlockObject(*bstrObjectId*, *bstrLockMode*, *blImmediateCommit*);

Parameter	Beschreibung
<i>bstrObjectId</i>	Das zu bearbeitende Objekt.
<i>bstrLockMode</i>	Der Modus des Bearbeitungszustandes einer Komponente. -> siehe Rückgabewerte für <i>bstrLockMode</i> .
<i>blImmediateCommit</i>	Ein Wert, der angibt, ob das Auslocken sofort durchgeführt werden soll. Wenn Sie es tun, können andere Benutzer die Komponente sofort editieren und müssen nicht auf Ihren nächsten <i>commit</i> warten.  <b>Hinweis:</b> Es wird empfohlen, diesen Parameter <b>nicht</b> mit '1' zu verwenden. <i>blImmediateCommit</i> <b>darf nicht</b> auf '1' gestellt werden, wenn ein Objekt tatsächlich geändert worden ist. Missbrauch kann zu gegenseitigen Datenbankblockierungen führen.

Die **Rückgabewerte** für *bstrLockMode* sind

- NOT\_DELETE
- NOT\_LINK
- READ
- LINK
- WRITE
- DELETE

#### Rückgabewert

<i>bFlag</i>	True(-1): Wenn das unlock Kommando ausgeführt wurde. False(0): Sonst (die Komponente kann z. B. nicht freigestellt werden).
--------------	--

#### Beispiel

#### Beispiel

```
sub main(id)
  base_id = Data.GetAttributeById(id, "relationobject2")
  ' Immediate commit of unlock
  iCommitUnlock = 0
  unlockit = CompLock.UnlockObject(base_id, "WRITE", iCommitUnlock)
  if unlockit = True then
    MsgBox("Unlock successful")
  else
    MsgBox("Unlock not successful. Object may not have been locked at all.")
  end if
end sub
```

### 3.12.1.3. UnlockAllObject

#### Syntax

```
UnlockAllObject(, blmmediateCommit);
```

Parameter	Beschreibung
blmmediateCommit	Ein Wert, der angibt, ob das Auslocken sofort durchgeführt werden soll. Wenn Sie es tun, können andere Benutzer die Komponente sofort editieren und müssen nicht auf Ihren nächsten <i>commit</i> warten.  In einem Dialog werden Sie gefragt, ob Sie den Unlock sofort vornehmen wollen.



**Hinweis:** Es wird empfohlen, diesen Parameter nicht mit '1' zu verwenden. Es ist schwierig sich einen Verwendungsfall vorzustellen, der die Verwendung der Option 'blmmediateCommit =1' rechtfertigt. blmmediateCommit darf nicht auf '1' gestellt werden, wenn ein Objekt tatsächlich geändert worden ist. Missbrauch kann zu gegenseitigen Datenbankblockierungen führen.

#### Rückgabewert

bFlag	True(-1): Wenn das unlock Kommando ausgeführt wurde. False(0): Sonst (die Komponente kann z. B. nicht freigestellt werden).
-------	--

#### Beispiel

#### Beispiel

```
Sub main(notused)
...
unlockit = CompLock.UnlockAllObject(0)
...
End sub
```



**Hinweis:** Die Verwendung ist ähnlich wie UnlockObject. Es wird davon ausgegangen, dass alle Objekte mit dem gleichen Lockmodus gelockt sind.

### 3.12.1.4. IsObjectLocked

#### SYNTAX

IsObjectLocked(bSTR bstrObjectId);

Parameter	Beschreibung
bstrObjectId	Die Objekt ID.

#### Rückgabewert

Rückgabewerte für *bFlag* sind

- True (-1): Wenn die Komponente bearbeitet wird
- False (0) Wenn die Komponente nicht bearbeitet wird

#### Beispiel

#### Beispiel

```
Sub main(id)
    base_id = Data.GetAttributeById(id, "relationobject2")
    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
```



---

**Hinweis:** Diese Methoden geben keine Auskunft über den Bearbeitungszustand eines Objektes. Wird das Objekt bearbeitet, können Sie über *GetObjectLockInfo* sich weitere Informationen anzeigen lassen.

---

### 3.12.1.5. GetObjectLockInfo

#### SYNTAX

GetObjectLockInfo(**BSTR** bstrObjectId);

Parameter	Beschreibung
<b>bstrObjectId</b>	Die Objekt ID.

Der zurückgegebene *plInfo* String enthält folgende Informationen (unterschieden nach CRLF):

- Lock Mode (z. B. "WRITE")
- User Name
- Computer Name
- Lock Date/Time

Im nachfolgenden Beispiel wird über eine Ausgabebox die Information angezeigt.

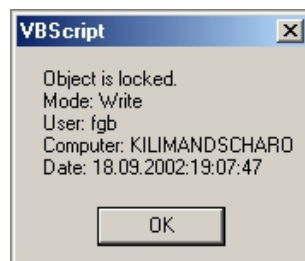
#### Rückgabewert

<b>plInfo</b>	Ein Sting der Informationen über den Bearbeitungszustand der Komponente enthält.
---------------	--

#### Beispiel

#### Beispiel

```
sub main(id)
base_id = Data.GetAttributeById(id, "relationobject2")
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
```



# Index

## A

Ausführen eines Skripts ..... 35

## B

Batch-Modus ..... 51, 52  
     Dateibasiert ..... 51  
     Objektbasiert ..... 52

## D

Data.CommitTransaction ..... 34

## E

eigene Transaktion ..... 33  
 Eigenschaften eines Skripts ..... 31  
     Parameter  
         Besitzberechtigung verwenden ..... 32  
         In eigener Transaktion ausführen ..... 32  
         Name ..... 31  
         Nummer ..... 31  
         Quelltext ..... 32  
         Skriptsprache ..... 32

## F

Fehlerbehandlung ..... 55  
     Laufzeitfehler ..... 55  
     Syntaxfehler ..... 55  
 Funktion ..... 55

## H

Haftungsausschluss ..... 3

## I

include Anweisung ..... 61  
 Interaktiven Modus ..... 36

## K

Klammersetzung ..... 58

## L

Ländereinstellungen ..... 19  
 Laufzeitfehler ..... 55

Liste aller Skriptaktionen ..... 71

## N

Neue Funktionen ..... 17  
     PE 5.10 ..... 21  
     PE 5.11 ..... 20  
     PE 5.12 ..... 19  
     PE 5.13 ..... 18  
     PE 5.14 ..... 17  
     PE 5.15 ..... 16  
     PE 5.16 ..... 15  
     PE 5.16SP4 ..... 14  
     PE 5.17 ..... 13  
     PE 5.18 ..... 13  
     PE 5.9 ..... 22

## O

Objekt-ID ..... 54

## P

Protokollfunktionen ..... 62  
 Prozedur ..... 54  
     Prozedur *main* ..... 55

## R

Reale Skriptaktionen ..... 39  
 Rechte im Skripting ..... 32  
 Rekursionen ..... 56

## S

ScriptItemConfig ..... 227  
 ScriptItemConvert ..... 236  
 ScriptItemGraphic ..... 168  
 ScriptItemList ..... 206  
 ScriptItemLock ..... 264  
 ScriptItemQuery ..... 209  
 ScriptItemRights ..... 143, 144  
     GetCurrentUser(PE 5.12) ..... 145  
     GetUserFullName ..... 144  
 ScriptItemUnit ..... 256  
 ScriptItemVersion ..... 239  
     CheckOut ..... 241  
     CheckOut ..... 243  
     CheckOutDeep ..... 242  
     CheckOutDeepSFI ..... 252  
     CheckOutSFI ..... 251  
     Create ..... 239  
     CreateDeep ..... 240  
     CreateReleaseTable ..... 248

CreateSFI .....	249, 250	Syntaxfehler .....	55
GetFirstVersion .....	247	<b>T</b>	
GetPlanningState .....	245	Transaktion .....	33
GetPlanningStates .....	246	<b>U</b>	
GetReleaseTable .....	249	Umwandeln VBS Skripte in VBS Makros .	67
PDXMLFileCreated .....	253	Unit .....	256
Promote .....	253	<b>V</b>	
SetPlanningState .....	244	Variable .....	55
UpdateRelations .....	255	VBA Objekt	
Use .....	254	Eigenschaften .....	69
Sicherheitsbedenken bei Einsatz von		VBA Objekt <i>Application</i> .....	68
Skripten .....	25	VBA-Host .....	63
Skript in einer eigenen Transaktion .....	33	VBA-Programme .....	63
Skriptaktion		VBA-Skripte .....	63
Attribute setzen .....	75	ausführen .....	66
Change Properties .....	84	erstellen .....	63
CloseProject .....	89	Umgebung .....	64, 65
DropProcessToBalancing .....	93	Umwandeln in VBS .....	67
Drucken(Print) .....	94, 95	Version .....	239
Init Properties .....	82	<b>W</b>	
Kind hinzufügen .....	74	Windows Scripting Host .....	24
Kopieren .....	77	<b>X</b>	
Neu .....	76	XScriptingHost Klassen .....	97
Neue Version .....	86		
Neues Kind .....	73		
OpenProject .....	88		
Planungsstatus setzen .....	87		
SaveBalancing .....	92		
StartBalancing .....	91		
Verknüpfen .....	78		
Verschieben .....	79		
Skriptaktionen			
Ereignisse .....	39		
im Kontextmenü .....	46, 48		
Parameter .....	37		
Skripte erstellt .....	30		
Skripte im Batch-Modus starten .....	51, 52		
Skripteditor .....	30		
Skriptzuweisungen .....	41		